Improved Pairwise Measurement-Based Surface Code

Linnea Grans-Samuelsson*[1], Ryan V. Mishmash*[1], David Aasen*[1], Christina Knapp[1], Bela Bauer[1], Brad Lackey[2], Marcus P. da Silva[2], and Parsa Bonderson*[1]

[1]Microsoft Station Q, Santa Barbara, California 93106-6105 USA
[2]Microsoft Quantum, Redmond, Washington 98052, USA

arXiv:2310.12981v2 [quant-ph] 23 Jul 2024

We devise a new realization of the surface code on a rectangular lattice of qubits utilizing single-qubit and nearest-neighbor two-qubit Pauli measurements and three auxiliary qubits per plaquette. This realization gains substantial advantages over prior pairwise measurement-based realizations of the surface code. It has a short operation period of 4 steps and our performance analysis for a standard circuit noise model yields a high fault-tolerance threshold of approximately 0.66%. The syndrome extraction circuits avoid bidirectional hook errors, so we can achieve full code distance by choosing appropriate boundary conditions. We also construct variants of the syndrome extraction circuits that entirely prevent hook errors, at the cost of larger circuit depth. This achieves full distance regardless of boundary conditions, with only a modest decrease in the threshold. Furthermore, we propose an efficient strategy for dealing with dead components (qubits and measurements) in our surface code realization, which can be adopted more generally for other surface code realizations. This new surface code realization is highly optimized for Majorana-based hardware, accounting for constraints imposed by layouts and the implementation of measurements, making it competitive with the recently proposed Floquet codes.

1 Introduction

The inherent fragility of quantum states has presented a formidable challenge in the pursuit of a scalable quantum computer. Quantum error correction will undoubtedly be essential in any practical realization. Due to its high threshold and local connectivity, the surface code [1, 2, 3] is a leading candidate for a scalable quantum error correcting code. Realizing a surface code subject to hardware constraints is a challenge, and different realizations will have varying performances. In particular, designing a syndrome extraction circuit composed of native operations, while maintaining code performance, is essential. In the broader context of the implementation of useful quantum algorithms, the resources required can be greatly impacted by the choice of code and how well matched it is to the hardware constraints [4, 5].

Most of the proposed implementations of the surface code in hardware have followed the CNOT gate-based realization of stabilizer measurement circuits of Ref. [3], or variants thereof. More recent proposals motivated by measurement-based Majorana quantum computing hardware [6] have

*Lead authors

considered pairwise measurement-based realizations of the surface code [7, 8, 9], which all utilized two auxiliary qubits for each bulk plaquette stabilizer measurement circuit. These pairwise measurement-based proposals each exhibited various significant drawbacks, such as complicated layouts and difficult measurements in Majorana hardware[1], relatively long circuits [8], or bidirectional hook errors [9]. The Floquet codes developed in Refs. [10, 11, 12, 5] provided a major advancement for the realization of pairwise measurement-based codes, largely eliminating such drawbacks and achieving better performance with a high fault-tolerance threshold.

In this paper, we devise and analyze a new implementation of the surface code using single- and two-qubit Pauli measurements on a rectangular array of qubits. This implementation utilizes three auxiliary qubits per stabilizer in the bulk. This property motivates us to refer to this surface code implementation as the "3aux" code when a short descriptive name is useful. Moreover, the stabilizer measurement circuits only utilize X, Z, XX, and ZZ Pauli measurements, with the pairwise measurements being directionally correlated between nearest-neighbor qubits, e.g. XX measurements are always between horizontal neighbors and ZZ are always between vertical neighbors. It is worth observing that our stabilizer measurement circuits and those of Refs. [7, 8, 9] are all closely related through circuit equivalences, for example using ZX-calculus relations (see e.g. Ref. [13] for a review of ZX-calculus). Despite these close relations, there are important physical differences that translate into implementation and performance advantages for our surface code realization. For example, our stabilizer measurement circuits can be pipelined in a manner that yields a minimal operation period of 4 steps for running the code, in which case no qubit is idle in any step.

The stabilizer measurement circuits in our surface code implementation have hook errors, similar to the original CNOT-based realizations discussed in Ref. [3]. Hook errors stem from single physical faults that, for a given circuit, are equivalent to higher-weight (e.g. two-qubit) errors on the data qubits (and not equivalent to single data qubit errors). These can reduce the *fault distance* [14], i.e., the minimum number of gate faults that causes an undetectable logical error; in particular, in the toric and surface codes, where logical operators are associated with a direction, hook errors reduce the fault distance when aligned with a logical operator of the same type. Unidirectionality or bidirectionality of hook errors indicate whether each type occurs in one or two directions with respect to the surface. In our case, the hook errors are unidirectional, with X and Z type hook errors in perpendicular directions. As such, our implementation can be utilized for the "rotated" surface code [15] without halving the fault distance by choosing boundary conditions and operation schedules accordingly, similar to the strategy used in Ref. [16]. This is in contrast to realizations with hook errors that cannot be favorably oriented, such as the bidirectional hook errors of Ref. [9]. Additionally, for situations where it is desirable, we can modify the circuit to prevent hook errors from occurring, at the expense of increasing the operation period to 7 steps. Given these properties, our code provides an interesting test bed for analyzing the effects of hook errors on code performance.

We can modify the bulk 4-gon (four data qubit) stabilizer measurement circuits to provide cir-

[1] In Majorana hardware, the double ancilla realizations [7, 8] require mixed tetron-hexon layouts and measurement loops involving coherent links; the windmill realization [8] requires measurement loops involving coherent links; and the pentagonal tiling realization [9] require measurement loops involving coherent links or long semiconductor segments.

cuits for measuring 1-gon, 2-gon, and 3-gon operators that interlock with the bulk 4-gon circuits. These n-gon measurement circuits utilize measurements from the same operation set, so constitute a minimal modification of the bulk. We can use the n-gons to implement code patches with any desired boundary conditions and alter the shape of code patches during operation, without disrupting the operation cycle. Moreover, we can utilize the n-gon measurement circuits to implement a protocol for dealing with dead components (i.e. dead qubits and pairwise measurements). In particular, we present an improved and maximally efficient variant of the protocol of Ref. [17] for dealing with dead components by measuring stabilizers for reduced n-gons that exclude dead components. Our protocol can be incorporated in a natural manner that requires minimal modification to the code operation, i.e. no addition to the set of measurements and no change to the bulk operation cycle. Furthermore, our circuit pipelining effectively alternates between Z-type and X-type n-gon measurements within each round, which allows all the "superplaquette" operators to be measured by measuring the reduced n-gons ("damaged plaquettes") within each round. We expect this to provide better performance than interleaved circuits would for such dead component protocols.

Since a motivation of this work was to develop code implementations for measurement-based Majorana quantum computing hardware, in particular in arrays of "tetron" qubits [6], our surface code realization is highly optimized for such hardware. In this regard, the rectangular lattice used for our code represents the simplest possible layout for such Majorana hardware. Moreover, for this hardware and layout, the measurements utilized in this code are extremely simple and likely to yield the lowest error rates of any set of physical Majorana parity measurements capable of generating a quantum error correcting code. Another consideration for this hardware is whether to use single or double columns of semiconductor "rails," which run between Majorana qubits to enable the measurements. Double-rail semiconductor layouts avoid physical conflicts between simultaneous measurements of adjacent qubits, generally allowing circuits to be implemented more efficiently in time. On the other hand, single-rail semiconductor layouts significantly simplify the requirements on fabrication and control for Majorana-based hardware, as compared to double-rail layouts. Using single-rail semiconductor layouts instead of double-rail generally increases the operation period by up to a factor of two; for example, using single-rail layouts instead of double-rail will double the operation period for the 4.8.8 Floquet code implementation in Majorana hardware described in Ref. [5]. We find that the use of single-rail layouts for our surface code realization can be implemented with a mild one step increase in the period.

Finally, we analyze the performance of our code for various scenarios. Our results indicate significant improvement compared to the previous pairwise measurement-based surface code realizations. In particular, simulating the logical failure rate for the standard circuit noise model, we find the fault-tolerance threshold to be approximately 0.66%, and achieve full distance sub-threshold scaling (when hook errors are appropriately addressed). Examining the effects of hook errors when using different boundary conditions and hook-preventing circuit modifications, we generally see the code performing better and achieving full distance in the deep sub-threshold regime for the scenarios where hook errors are benign or absent. Interestingly, the threshold and near threshold behavior are only modestly reduced by the hook-preventing modifications, in contrast to hook-flagging modifications of CNOT-based circuits [18]. Moreover, we compare our code to the state-of-the-art pairwise measurement-based 4.8.8 Floquet code [10, 11, 5], and find it to be reasonably competitive, especially at lower physical error rates. For variants of these two codes that are compatible

protocol	qubit count	depth	threshold	Majorana hardware
3aux	$O(4d_f^2)$	4	0.66%	simple
3aux, single-rail	$O(4d_f^2)$	5	0.51%	simple
double ancilla	$O(3d_f^2)$	10	0.24%	complicated
windmill	$O(2d_f^2)$	20	0.15%	complicated
pentagonal	$O(12d_f^2)$	6	0.4% *	complicated
4.8.8	$O(4d_f^2)$	3	1.3%	simple
honeycomb	$O(6d_f^2)$	3	1.3%	simple
4.8.8, single-rail	$O(4d_f^2)$	6	0.52%	simple

Table 1: Comparison between key properties of pairwise measurement-based codes, including our realization of the surface code (denoted "3aux"), the double ancilla and windmill realizations of Ref. [8], the pentagonal tiling realization of Ref. [9], and the Floquet code on the 4.8.8 and honeycomb lattices [10, 11, 12, 5]. The term "single-rail" indicates variants of the corresponding codes that are needed to make them compatible with Majorana hardware using single-rail semiconductor layouts. Total qubit count for a logical patch is given to leading order as function of fault distance d_f. Circuit depth is given per round of syndrome extraction. Fault tolerance thresholds are computed with respect to the noise model of Ref. [8], except for that of the pentagonal tiling realization from Ref. [9] (indicated by *), which uses a slightly different noise model. We indicate whether the code can be implemented in Majorana hardware using simple layouts and measurements, or if it requires complicated layouts and measurements that are likely prohibitive.

with implementation in Majorana hardware with single-rail layouts, their performance becomes even more competitive and their thresholds nearly identical. We provide a summary comparison of key properties of the various pairwise measurement-based codes in Table 1. (For the detailed comparison of performance and resource requirements between our surface code realizations and the 4.8.8 Floquet code, see Sec. 7.3.)

The structure of this paper is as follows. In Sec. 2, we introduce our code, presenting the pairwise measurement-based stabilizer measurement circuits and pipelining to realize the surface code on a rectangular lattice. In Sec. 3, we describe the occurrence of hook errors in our circuits and present modifications of our measurement circuits that prevent the occurrence of hook errors. In Sec. 4, we detail the measurement circuits for all n-gons, with $n = 1, 2, 3$, and utilize these for surface code patches with boundaries. In Sec. 5, we describe our proposed strategy for dealing with dead components and its application to our surface code realization. In Sec. 6, we discuss the implementation in Majorana hardware and provide a modification of the pipelining to make the code compatible with single-rail semiconductor layouts. In Sec. 7, we describe our numerical simulations of code performance and resource estimation and present the results.

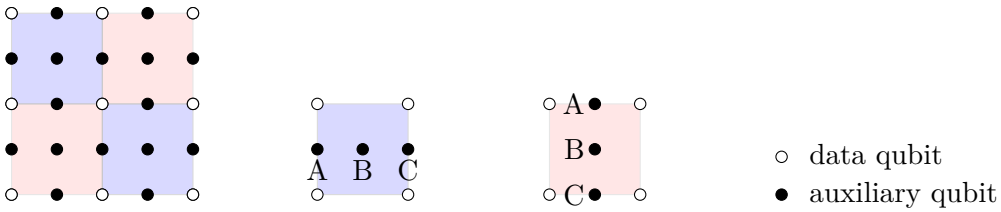


Figure 1: A rectangular lattice of qubits used for a pairwise measurement-based realization of the surface code. Data qubits are shown as open dots and auxiliary qubits are shown as solid dots. The blue and red squares will correspond to Z-type and X-type plaquettes (4-gons), respectively. Each plaquette exclusively utilizes three auxiliary qubits (labeled A-C) to execute its stabilizer measurement circuit.

2 The Code Circuits

We use a rectangular lattice of qubits to implement a pairwise measurement-based realization of the surface code, as shown in Fig. 1. The plaquettes correspond to 4-gon stabilizer measurements, which are arranged in a checkerboard pattern of Z-type ($ZZZZ$ stabilizers) and X-type ($XXXX$ stabilizers). Each plaquette exclusively utilizes three auxiliary qubits to perform the stabilizer measurement circuit.

In Fig. 2, we present a circuit diagram for M_{ZZZZ}, the measurement of $ZZZZ$ on four data qubits, using three auxiliary qubits. The $ZZZZ$ stabilizer measurement outcome is given by the product of the measurement outcomes of the six Z-type measurements, i.e. M_{Z_B} (initial), $M_{Z_AZ_1}$, $M_{Z_CZ_2}$, $M_{Z_AZ_3}$, $M_{Z_CZ_4}$, and M_{Z_B} (final). There are various methods for verify the functioning of this circuit. One straightforward method is to compute the instantaneous stabilizer group (ISG) [10] following each of the six steps of the circuit:

$$\begin{aligned}
0Z: &\quad \langle X_A\rangle \\
1Z: &\quad \langle Z_1Z_A, Z_B, X_C\rangle \\
2Z: &\quad \langle X_AX_B, Z_CZ_2, Z_1Z_AZ_B\rangle \\
3Z: &\quad \langle Z_3Z_A, X_BX_C, Z_1Z_AZ_BZ_CZ_2\rangle \\
4Z: &\quad \langle X_A, Z_B, Z_CZ_4, Z_1Z_BZ_CZ_2Z_3\rangle \\
5Z: &\quad \langle X_C, X_A, Z_B, Z_1Z_2Z_3Z_4\rangle\,.
\end{aligned} \tag{1}$$

Alternatively, one may verify the functioning of the circuit through a straightforward application of ZX-calculus [13].

By replacing $X \leftrightarrow Z$ in this circuit, we obtain the circuit for measuring $XXXX$ on four data qubits shown in Fig. 3. We note that there are numerous equivalences that can be applied to produce equivalent stabilizer circuits. For example, one could apply basis changes, permute data or auxiliary qubit labels, or modify the operation schedule, e.g. by adding/removing time steps and sliding measurements into different time steps (without sliding past other measurements on the same qubit lines). We will use this flexibility to incorporate certain appealing features in the implementation.

One desirable feature is to minimize the operation time. In order to compress the circuit into the shortest possible operation period, we can use the measurement schedules shown for M_{ZZZZ}

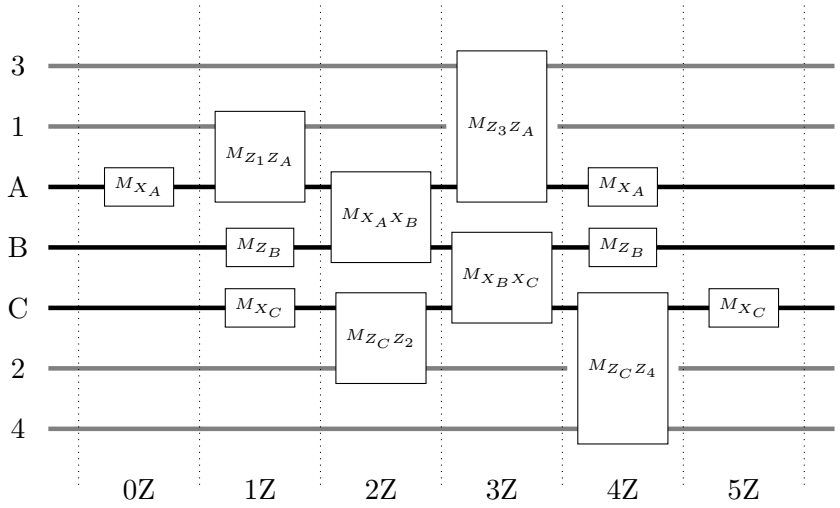


Figure 2: The M_{ZZZZ} circuit for measuring $ZZZZ$ on four data qubits using three auxiliary qubits. The data qubits are labeled 1-4 in the order that they are addressed in this circuit. The auxiliary qubits are labeled A-C. The measurement schedule shown here is useful for repeatedly applying the measurement circuit with a four step period, which can be done by repeating steps 1-4.

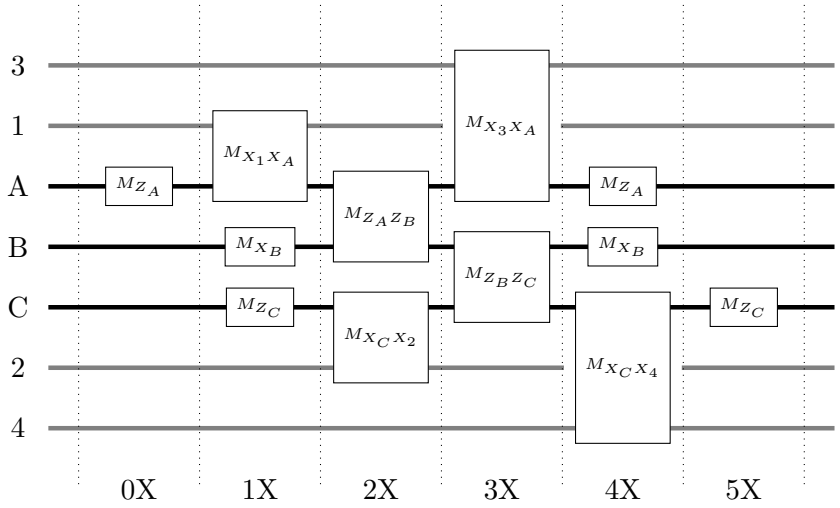


Figure 3: The M_{XXXX} circuit for measuring $XXXX$ on four data qubits using three auxiliary qubits may be obtained from M_{ZZZZ} by interchanging $X \leftrightarrow Z$ for all qubits in the circuit.

and M_{XXXX} in Figs. 2 and 3. While these circuits are shown with six steps, if we are repeating a stabilizer measurement circuit, the single-qubit measurements on auxiliary qubits can serve as both the final measurement of one round of running the stabilizer measurement circuit and the initial measurement of the subsequent round. Thus, we can avoid repeating the single-qubit measurements on auxiliary qubits A and C between every round, removing steps 0 and 5 from all but the very first and last rounds of applying the circuit, respectively. In this way, the number of steps necessary for applying r rounds of the stabilizer measurement circuit is $4r+2$, i.e. the circuit can be implemented with a 4 step period. Here, the initial and final rounds correspond to the initial measurement of

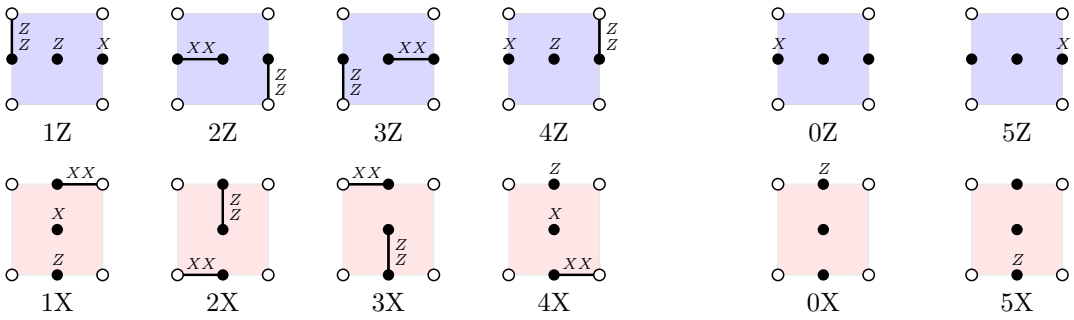


Figure 4: The M_{ZZZZ} and M_{XXXX} measurement circuit steps for Z-type and X-type plaquettes. Labels indicate the operators measured on the respective qubits in the given step, with connecting lines indicating pairwise measurement. These circuits can be applied in parallel with a relative shift in their operation schedules. A 4 step period for repeated application of these surface code stabilizer measurement circuits is obtained using the pipelining: $\cdots, (1Z, 3X), (2Z, 4X), (3Z, 1X), (4Z, 2X), \cdots$. Steps 1-4 of a given circuit are applied repeatedly, whereas steps 0 and 5 are only used in the ramp up and down of the repetition of circuits, as indicated in Eqs. (2)-(4).

qubit A (step 0) and final measurement of qubit C (step 5), while steps 1-4 are repeated r times in between. We will find it useful to repeat the single-qubit measurement of auxiliary B, as readout errors of that measurement would otherwise affect the 4-gon stabilizer readout of the two successive stabilizer measurements. This single-qubit measurement can be repeated without slowing down the circuit by incorporating it in steps 1 and 4.

Next, we consider applying these M_{ZZZZ} and M_{XXXX} circuits on the rectangular lattice with a checkerboard pattern in order to implement the surface code. (For now, we focus on the bulk plaquettes and will return to the matter of boundaries and smaller n-gons in Sec. 4.) When designing this implementation, we must ensure that operations in neighboring plaquettes do not conflict with each other. Specifically, the measurement sequences should avoid multiply-addressing any given qubit at any step. (One should also avoid conflicting uses of measurement components in hardware layouts where this may occur; we will consider this for Majorana hardware in Sec. 6.) Furthermore, the operation schedule must coordinate between adjacent plaquettes in a manner that correctly builds up the instantaneous stabilizer group to yield the desired M_{ZZZZ} and M_{XXXX} measurements needed for a surface code.

We can achieve the necessary properties by carefully pipelining the circuits with the data qubits addressed in the manner shown in Fig. 4, with the steps pipelined as

$$\cdots, (1Z, 3X), (2Z, 4X), (3Z, 1X), (4Z, 2X), \cdots . \tag{2}$$

In this way, applying r rounds of the surface code stabilizer measurements takes $4r + 4$ steps, i.e. has period 4, where we ramp up with the steps

$$(0Z, -), (1Z, -), (2Z, 0X), (3Z, 1X), \cdots \tag{3}$$

and ramp down with the steps

$$\cdots, (4Z, 2X), (5Z, 3X), (-, 4X), (-, 5X). \tag{4}$$

Alternatively, we could interchange X and Z in the ramp up and down. With the pipelining in Eq. (2), the value of the $ZZZZ$ stabilizer between time steps $(2Z, 4X)$ and $(3Z, 1X)$ is measured by the M_{ZZZZ} circuit, and similarly the value of the $XXXX$ stabilizer between time steps $(4Z, 2X)$ and $(1Z, 3X)$ is measured by the M_{XXXX} circuit. The reason for specifying the time steps for the data qubit stabilizers is because, when considering the full code, the data qubit operators at different times will generally not be equivalent due to other plaquettes' stabilizer circuits addressing those data qubits.

We note that it is also possible to correctly implement the M_{ZZZZ} and M_{XXXX} circuits by interleaving them on a synchronous schedule by appropriately choosing the order in which the circuit addresses the data qubits, the details of which are given in Appendix A. This turns out to have significant disadvantages compared to the pipelined measurement schedule presented above, so we do not focus on it in this paper.

3 Hook Errors and their Prevention

The stabilizer measurement circuits of Figs. 2 and 3 exhibit hook errors, which are errors that stem from a single fault, but that are equivalent to two-qubit errors on the data qubits. In this paper, we distinguish between faults and errors as in Ref. [5]: a fault is a failure of a circuit component (in our case, a measurement or an idling qubit) which results in a set of errors, and an error is represented by a single-qubit Pauli operator applied after the circuit component or a flipped measurement outcome, i.e. a readout error. For example, a fault on a two-qubit measurement can result in a readout error and a Pauli error on either or both qubits involved. Hook errors are concerning because they can reduce the fault distance and harm performance. We use "code distance" d to mean the minimal number of single data qubit errors that combine to produce a logical error, and "fault distance" d_{f} to mean the minimal number of faults such that the resulting errors combine to produce a logical error. The fault distance [14] is the effective distance achieved by a particular realization of the code, which depends on the specific details of the circuits. In some situations, the directionality of the hook errors allows them to be oriented perpendicular to the corresponding logical operators through a judicious choice of boundary conditions, thereby making their effect on the encoded logical state benign [16]. We now discuss how hook errors occur in the stabilizer measurement circuits presented in Sec. 2. We find that they are unidirectional and can be made benign, e.g. for a rotated surface code patch, using an appropriate choice of boundary conditions. Moreover, we discuss how to modify these circuits to prevent hook errors altogether.

For the stabilizer measurement circuits of Figs. 2 and 3, readout errors and two-qubit errors stemming from faults on the pairwise measurements of auxiliary qubits in our circuits are equivalent to two-qubit errors on data qubits. In more detail for the M_{ZZZZ} circuit, a Pauli error Z_B between steps 2 and 3 is equivalent to a Z_1Z_3 or Z_2Z_4 error on the data qubits, as shown in Fig. 5.[2] A readout error stemming from a fault on the $M_{X_AX_B}$ or $M_{X_BX_C}$ measurement is also equivalent to a Z_1Z_3 or Z_2Z_4 error on the data qubits. Similarly, for the M_{XXXX} circuit, a Pauli error X_B between steps 2 and 3 or a readout error at the $M_{Z_AZ_B}$ or $M_{Z_BZ_C}$ measurement is equivalent to a X_1X_3

[2]When the stabilizer circuits are pipelined as in Eq. (2), this hook error is equivalent to a data ZZ error occurring at that same time interval (between steps 2 and 3). This ZZ error on data qubits is not necessarily equivalent to a ZZ error at a different time step, since the neighboring plaquettes' M_{XXXX} circuits also address these data qubits.

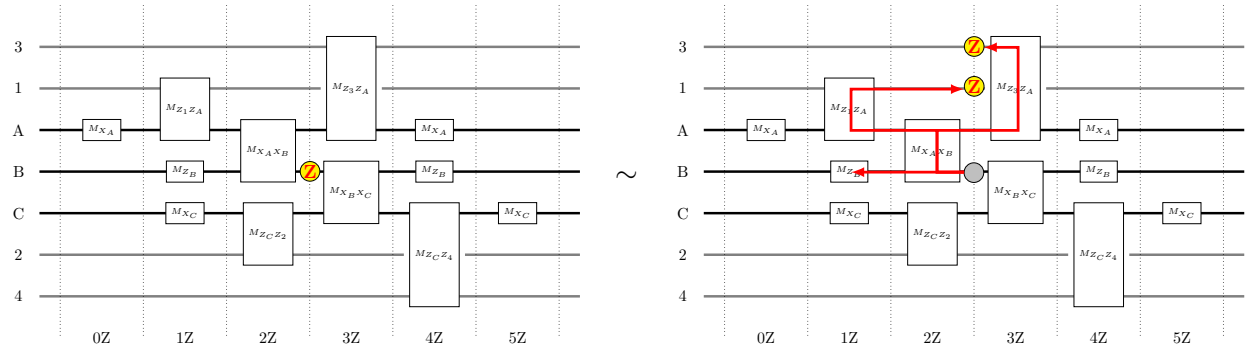


Figure 5: An example of a hook error in the M_{ZZZZ} measurement circuit is given by a Z error on auxiliary qubit B that occurs between steps 2 and 3, as shown in the circuit on the left. This error is equivalent to a ZZ error on data qubits 1 and 3, as shown in the circuit on the right. (Errors are marked in yellow. Red lines indicate the path through which the Pauli operator is "pushed" through the circuit elements using equivalences.)

or X_2X_4 error on the data qubits. If we do not repeat the single-qubit measurements on auxiliary qubits A and C, then readout errors stemming from faults on these non-repeated measurements would be equivalent to these same two-qubit errors on data qubits listed above for the respective circuit types.

We observe that the hook errors in our circuits for a given plaquette type are unidirectional, with the direction in our code realization correlated with the error type: the Z-plaquettes' hook errors correspond to ZZ data qubit errors in the vertical direction and the X-plaquettes' hook errors correspond to XX data qubit errors in the horizontal direction. This is a useful property that, for example, allows us to choose boundary conditions for the "rotated" surface code on a planar patch such that the logical operators are aligned perpendicular to the corresponding type of hook errors; that is, the logical qubit's Z-logical string operators are horizontal and the X-logical string operators are vertical. This choice prevents the hook errors from reducing the fault distance of the code (at least during logical idle). We will return to this matter in Sec. 4, after describing the circuits for boundary stabilizer (3-gon, 2-gon, and 1-gon) measurements.

Another way we can address the hook errors is to modify our stabilizer measurement circuits so that they detect and distinguish the occurrence of these problematic faults, and prevent the resulting errors from being equivalent to two data qubit errors. This strategy could be useful in scenarios where the logical qubit operators are not (or cannot be) aligned strictly perpendicular to the direction of their corresponding hook errors, such as when performing certain logical gate operations (see, e.g., Ref. [14]). One way of modifying our circuits to prevent the hook errors is to repeat the measurements with which they are associated. For the pairwise auxiliary qubit measurements, immediately repeating each measurement does not fix the problem, but repeating the pair of measurements (alternating between the AB and BC measurements) does. With these modifications, the resulting hook-preventing circuit for M_{ZZZZ} is shown in Fig. 6. The repeated measurements add new "detectors" to the circuit, which allow us to distinguish the errors stemming from the problematic faults from the previously equivalent two data qubit errors. (A detector is a set of measurements for which the product (or parity) of their outcomes is fixed in the absence of errors [19]; see Sec. 7 for further discussion.) The M_{XXXX} circuit can be obtained from this circuit

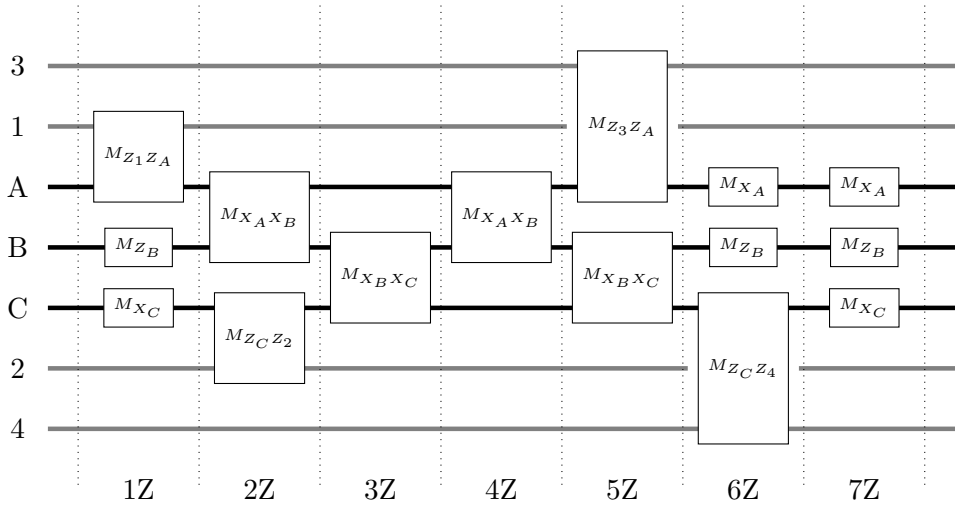


Figure 6: The modified circuit for M_{ZZZZ} in which there are no hook errors. Steps 3, 4, and 7 shown here are the additional steps that have been inserted into the original circuit from Fig. 2. (The M_{Z_B} auxiliary qubit measurement only needs to occur twice per cycle, but we have shown it occurring three times.) Here, we only show the steps for one round of repeated application of the measurement circuit. In order to ramp up the circuit, one needs to begin with a step 0 that applies a M_{X_A} measurement before step 1; this could be achieved (with additional redundant measurements) by applying step 7.

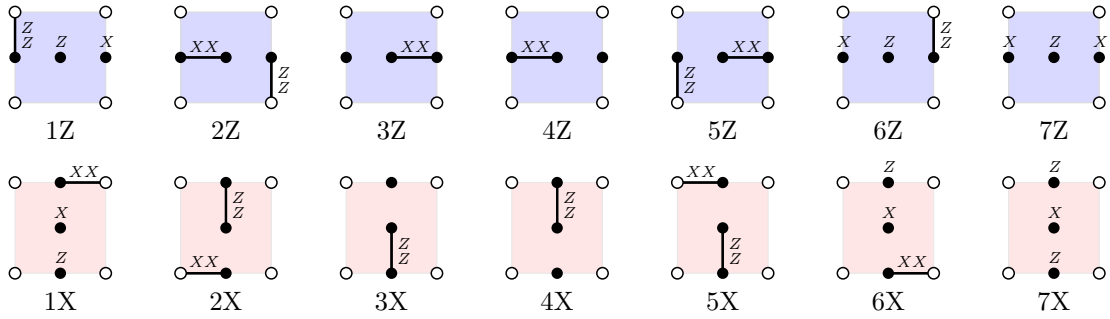


Figure 7: The hook-preventing M_{ZZZZ} and M_{XXXX} measurement circuit steps for Z-type and X-type plaquettes. These can be pipelined as in Eq. (5) to achieve a 7 step period.

by replacing $X \leftrightarrow Z$. Using these hook-preventing stabilizer measurement circuits, the surface code can be implemented with a 7-step period. Using the same order of addressing data qubits as used for the unmodified circuits, as shown in Fig. 7, we find several options for pipelining the circuits to achieve this periodicity, namely:

$$\begin{aligned}
1: &\quad \cdots, (1Z, 6X), (2Z, 7X), (3Z, 1X), (4Z, 2X), (5Z, 3X), (6Z, 4X), (7Z, 5X), \cdots \\
2: &\quad \cdots, (1Z, 5X), (2Z, 6X), (3Z, 7X), (4Z, 1X), (5Z, 2X), (6Z, 3X), (7Z, 4X), \cdots \\
3: &\quad \cdots, (1Z, 4X), (2Z, 5X), (3Z, 6X), (4Z, 7X), (5Z, 1X), (6Z, 2X), (7Z, 3X), \cdots \\
4: &\quad \cdots, (1Z, 3X), (2Z, 4X), (3Z, 5X), (4Z, 6X), (5Z, 7X), (6Z, 1X), (7Z, 2X), \cdots
\end{aligned} \tag{5}$$

It it worth mentioning that this hook-preventing method of repeating the problematic measurements can be applied to other measurement-based code implementations, such as the pentagonal tiling realization of the surface code devised in Ref. [9]. We present details for this example in Appendix B.

4 Smaller n-gons and Boundaries

In order to operate the code on surfaces with a boundary, we need circuits for measuring the multi-qubit Pauli operators of 1, 2, or 3 data qubits, which we call 1-gons, 2-gons, or 3-gons, respectively. A convenient way to produce stabilizer measurement circuits for these boundary stabilizers is to start with the corresponding 4-gon measurement circuits and remove the appropriate data qubits. In doing so, we may reduce or remove certain measurements from the circuit, where reduction changes a two-qubit measurement involving a removed data qubit into a single-qubit measurement of the same type on the corresponding auxiliary qubit, e.g. $M_{Z_A Z_1} \to M_{Z_A}$. We may also remove auxiliary qubits from the n-gon when removing data qubits in this way, depending on which type of n-gon we are producing. In particular, 2-gons with data qubits addressed by a common auxiliary qubit only require that auxiliary qubit (the other two can be removed), and 1-gons require no auxiliary qubits. The resulting n-gon measurement circuits are shown in Fig. 8. (The ramp up and down steps 0 and 5 are not shown; we can simply use those of the 4-gon circuits when the step is required.) A convenient property of producing n-gon measurement circuits in this manner is that they interlock with the bulk 4-gon circuits. In other words, for a system that uses these n-gon measurement circuits, we can operate all Z-type n-gons on the same schedule and all X-type n-gons on the same schedule, without further modification, e.g. we can use the pipelining in Eq. (2) of Z- and X-plaquette circuits for all the n-gons.

With these n-gon measurement circuits, it is straightforward to define the code on surfaces with boundaries. However, an issue that arises is that certain choices of boundary conditions may be more advantageous than others. For example, when putting the surface code on a square patch of data qubits with the "rotated surface code" boundary conditions [15], there are two options for how to assign boundary types to the edges of the square patch. The good choice for our code realization, shown in Fig. 9(a), corresponding to using Z-type 2-gons along vertical edges and X-type 2-gons along horizontal edges. This choice aligns the Z and X logical string operators perpendicular to the direction of the Z- and X-plaquettes' hook errors, respectively. The bad choice for our surface code realization, shown in Fig. 9(b), corresponding to using Z-type 2-gons along horizontal edges and X-type 2-gons along vertical edges. This choice utilizes more auxiliary qubits at the boundary and, more importantly, aligns the logical string operators with the corresponding hook errors, which halves the fault distance of the code. In contrast, the original (unrotated) surface code boundary conditions [2] is implemented using 3-gons along the boundary edges, with Z-type and X-type 3-gons corresponding to "rough" and "smooth" boundaries. In this case, the logical string operators are aligned diagonally across the plaquettes, so the hook errors do not affect the code distance. However, the relation between distance and number of qubits for the original surface code is worse than that of the (good) rotated surface code due to this diagonal alignment of logical string operators with respect to the plaquettes.

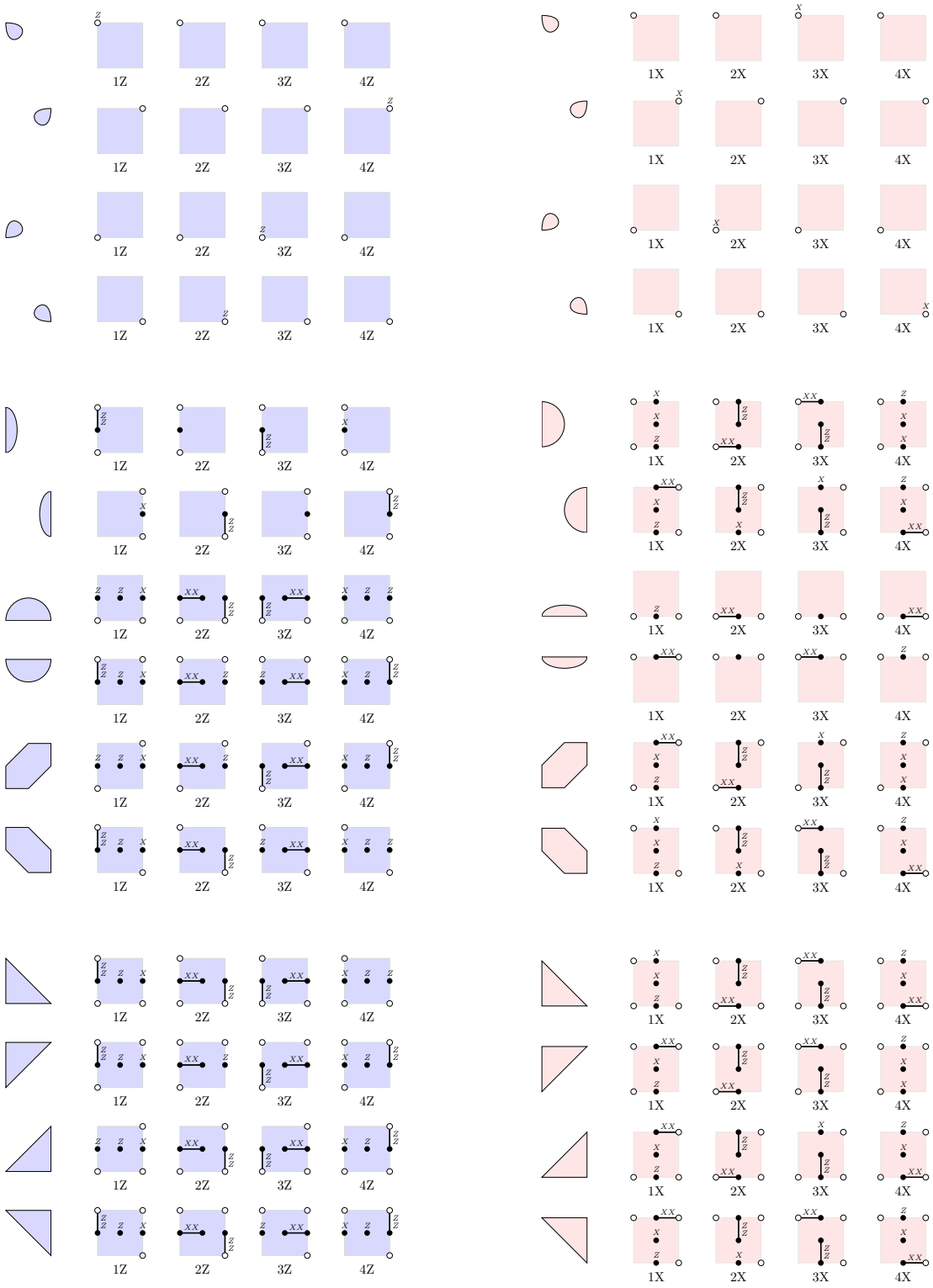


Figure 8: Z-type (left) and X-type (right) measurement circuits for 1-gons, 2-gons, and 3-gons.

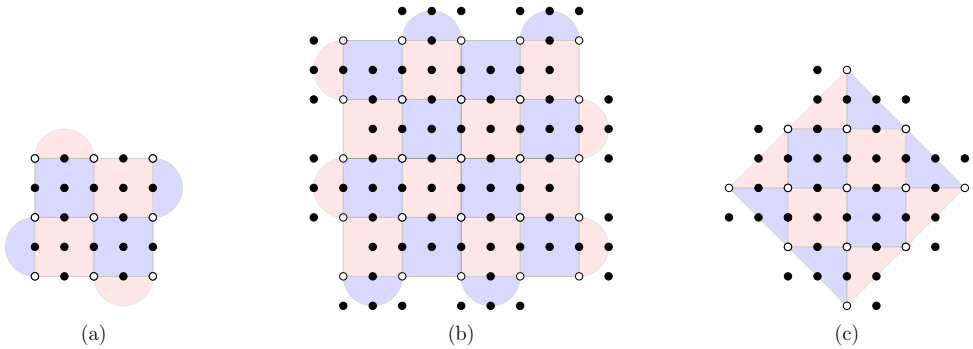


Figure 9: Different boundary conditions for a square patch of surface code, all shown for fault distance $d_{\mathrm{f}} = 3$ when using the period 4 circuits of Figs. 4 and 8. (a) The rotated surface code with a good choice of boundary conditions aligns the logical strings perpendicular to the corresponding hook errors of the same type, so $d_{\mathrm{f}} = d$. The relation between number of physical qubits and distance in this case is $N = 4d^2 - 4d + 1$. (b) The rotated surface code with a bad choice of boundary conditions aligns the logical strings parallel to the corresponding hook errors of the same type, which halves the fault distance, so $d_{\mathrm{f}} = \lceil d/2 \rceil$. The resulting relation between number of physical qubits and the distance in this case is $N = 4d^2 - 3$, giving the scaling with fault distance of $N = O(16d_{\mathrm{f}}^2)$. (Using the period 7 hook-preventing circuits of Fig. 7 would avoid halving the distance.) (c) The original (unrotated) surface code has the logical strings aligned diagonal to the direction of the hook errors, so $d_{\mathrm{f}} = d$. The relation between number of physical qubits and distance in this case is $N = 8d^2 - 8d + 1$.

5 Fault Tolerance in the Presence of Dead Components

An important problem to address when implementing error-correcting codes in physical hardware is maintaining fault tolerance in the presence of physical components that are nonfunctional or exhibit substantially higher error rates than most of the components. We can map the effects of faulty physical components to an effective computational model, where they are specified in terms of qubits, computational gates, and measurements. For our purposes, we will refer to qubits, computational gates, and measurements that are nonfunctional or exhibiting atypically higher error rates as "dead components." The dead components can potentially be identified during the bring-up and calibration phase of operating the hardware (if dead at the time) or also during the operation of the error-correcting code when error syndromes indicate a qubit or operation is exhibiting a high error rate.

Ref. [20] introduced a strategy for dealing with dead data qubits by removing them from the code operation and forming "superplaquette" operators, which are products of the original plaquette operators (of the same type) that exclude the dead qubits.[3] Building on this idea, Ref. [17] proposed to generate the superplaquette measurements by measuring the "damaged stabilizers," i.e. the original plaquette operators reduced by removal of the dead data qubits, in a manner such that they combine to yield all the superplaquette stabilizers. In general, a damaged stabilizer will not commute with all other stabilizers, so they cannot all be simultaneously measured. In light of this, Ref. [17] proposed to deal with dead components by successively measuring the damaged stabilizers of Z-type and X-type in alternating rounds of stabilizer measurements, while continuing

[3] We refer to all the stabilizers of the surface code as plaquette operators, distinguishing them as Z-type and X-type, rather than "plaquette" and "star" operators, respectively.

to measure both types of undamaged stabilizers every round. In this way, damaged stabilizers would be measured half as often as the undamaged stabilizers. Using this alternation between measuring Z-type and X-type damaged stabilizers, the instantaneous stabilizer group after a given round includes the damaged stabilizers of the type from that round, together with the superplaquette stabilizers of the other type formed from the previous round's damaged stabilizers (but not the previous round's individual damaged stabilizers).

As a concrete realization, Ref. [17] considered the CNOT-based implementation of plaquette stabilizer measurements, using an auxiliary qubit for each plaquette and CNOT gates connecting auxiliary to data qubits, with a suitable interleaving of the circuits of neighboring plaquettes. For a dead data qubit, the CNOT gates involving that qubit are simply removed from the respective stabilizer measurement circuits. For these circuits on damaged plaquettes, Ref. [17] stated that the normal circuit schedule could not be used because it would randomize the superplaquette values due to the anti-commutation of damaged stabilizers, and they consequently required measurement circuits for Z-type and X-type damaged stabilizers to be applied in separate alternating rounds. This claim is incorrect; the normal schedule for the interleaved CNOT-based implementation can, in fact, be utilized for the damaged stabilizers. [4] One can understand this, at least qualitatively, by using circuit equivalences, e.g. via ZX-calculus, and deforming the circuits so that damaged plaquette measurements appear as layered (in fictitious time), rather than interleaved. In this manner, we see that the CNOT-based stabilizer measurements are not actually simultaneous and that the order in which data qubits are addressed by the interleaved circuits determines an effective order (the layering order in fictitious time) in which the damaged stabilizers are measured (Ref. [17] observed this property in their Fig. 11). Moreover, each layer of the effective ordering contains stabilizers of either only Z-type or only X-type, but not both. Tracking the corresponding effective instantaneous stabilizer group with respect to this effective measurement order reveals that the superplaquette stabilizers are formed over multiple rounds of applying the measurement circuits. Attempting to form the superplaquette stabilizers from the damaged plaquette measurements of a single round would indeed yield randomized values, but appropriate compositions of pieces from multiple rounds produce values for superplaquette operators that are deterministic in the absence of errors. The number of rounds across which these superplaquette stabilizers are composed can grow with the size of a connected region of dead components. In particular, a region with l effective measurement layers for the damaged stabilizers in each round will require $\lceil (l-1)/2 \rceil$ and $\lceil l/2 \rceil$ to build up the two respective types of superplaquette stabilizers. For example, a region with a single dead data qubit has $l = 3$, a region with two adjacent dead data qubits has $l = 4$, and a region with a smallest square of dead data qubits (a dead plaquette) has $l = 5$. (An undamaged region can be thought of as having $l = 2$.) Pulling this understanding back to the interleaved circuits running with the normal circuit schedule, we see that damaged plaquette stabilizers would be measured at the same rate as undamaged stabilizers, but this would yield a reduced number of superplaquette measurements due to finite time. In particular, r rounds of measurement circuits would respectively yield $r + 1 - \lceil (l-1)/2 \rceil$ and $r + 1 - \lceil l/2 \rceil$ measurements of the two types of superplaquette stabilizers. We typically expect to be operating in a regime where $l < d_{\mathrm{f}}$ and $r \approx d_{\mathrm{f}}$, in which case this circuit schedule provides a greater number of superplaquette stabilizer measurements and

[4] We thank Reviewer 1 for bringing this to our attention.

detectors than alternating rounds between measuring Z-type and X-type damaged stabilizers. As such, we expect operating the circuits this way to provide better performance in the presence of dead components. Interleaved versions of our pairwise measurement-based stabilizer circuits, as described in Appendix A, behavior similarly with respect to effective time ordering of damaged plaquettes and forming superplaquette stabilizers over multiple rounds. Again, for r rounds of measurement circuits, this will yield $r+1-\lceil (l-1)/2 \rceil$ and $r+1-\lceil l/2 \rceil$ measurements of the two types of superplaquette stabilizers.

In contrast, the pipelining we use for our measurement-based circuits, e.g. in Eq. (2), effectively operates as alternation between all Z-type and all X-type n-gon measurements. Again, this can be seen by tracking when the data qubits of neighboring plaquettes are addressed by the circuits for the different plaquette types, which allows one to isolate each plaquette's circuit in fictitious time. Regardless of how many components are dead, there will be exactly two layers in the effective ordering in each round. Thus, r rounds of pipelined measurement circuits yields r measurements for each of the two types of superplaquette stabilizers. In light of this, we expect circuits that are pipelined in this manner to provide better performance in the presence of dead components than interleaved circuits. We note that a similar pipelining can be applied to the CNOT-based implementation of stabilizer measurements, which would yield the same advantage, though with the potential drawback that the auxiliary qubit initialization and measurement steps for the Z-type and X-type stabilizers would not be concurrent, potentially complicating timing of physical operations.

Returning to the situation where all undamaged stabilizers are measured in each measurement round, but only one type of the damaged stabilizers are measured, Ref. [21] found improved performance for larger regions of dead components by modifying the protocol of Ref. [17] to alternate between l repeated rounds of damaged stabilizer measurements of each type, where l is the linear size of the dead region. For our pipelined circuits, we could follow a similar protocol of repeating damaged stabilizer measurements of each type l times before alternating, but this would require halving the rate at which damaged stabilizers are measured. It is not obvious whether the trade-off for employing this strategy would improve performance in our case; in fact, we expect it to decrease performance, since it would require halving the rate at which damaged plaquettes are measured.

Ref. [17] additionally described a strategy for dealing with dead auxiliary qubits and CNOT gates (which they called syndrome qubit and link fabrication errors, respectively). For this, they identify all the data qubits directly interacting with a dead CNOT gate or auxiliary qubit, i.e. the data qubit acted upon by a given CNOT gate or all data qubits in the plaquette associated with the given auxiliary qubit, respectively. Then, all such data qubits (even though not dead) are removed from the code, together with all the dead components.

We propose a different strategy for dealing with the dead auxiliary qubits and connections that avoids unnecessarily removing data qubits that are not dead, and which we expect should improve performance. Here, "connection" refers to a multi-qubit operation, which may include computational gates (e.g. CNOTs) or measurements (e.g. pairwise Pauli measurements) acting on multiple physical qubits. Our strategy for determining the modification of the code operation to remove dead components can be divided into three steps:

1. For any n-gon involving m dead data qubits, reduce it to a $(n-m)$-gon by removing the dead data qubits.

2. For any n-gon (possibly the result of a reduction in step 1) involving dead auxiliary qubits, split it up into a n_1-gon, ... , and n_k-gon, where $n_1 + \cdots + n_k = n$, such that k is minimized and none of the resulting measurement circuits utilize the dead auxiliary qubits.

3. For any n-gon (possibly the result of a reduction and/or splitting in steps 1 and 2) involving dead connections, split it up into a n_1-gon, ... , and n_k-gon, where $n_1 + \cdots + n_k = n$, such that k is minimized and none of the resulting measurement circuits utilize the dead connections.

The result of applying each step is unique, i.e. no choices need to be made (at least for simple enough realizations, including all those of interest examined in this paper). Within each step, the reduction or splitting of a n-gon can be determined iteratively by assessing one component at a time, until all dead components of that step's type have been removed. Splitting n-gons (steps 2 and 3) always preserves the number of data qubits. On the other hand, the reductions and splittings (all steps) may result in auxiliary qubits or connections that are not dead, but are no longer utilized in the stabilizer measurement circuits; such auxiliary qubits and connections are collaterally removed from the code operation, as they no longer participate. For example, if an auxiliary qubit is dead, then all connections to it are removed; similarly, if all connections to a qubit are dead, that qubit is removed. We will see more examples of this when we consider specific realizations. In examples (not considered explicitly in this paper) where certain auxiliary qubits or connections are shared by multiple plaquettes' stabilizer measurement circuits, they may, in principle, be dead or removed with respect to one plaquette, but not another. For these situations, the functionality of the components may be assessed for each plaquette, which can then be independently modified accordingly. Though it would not change the functioning of the code, if there are regions that become completely disconnected from the main region of the code, e.g. a single data qubit with no connections to other qubits, these can safely be removed as well.

This protocol provides the most efficient salvaging of functioning components (without adding components or elements to the set of operations) when removing the dead components – no functioning data qubits are removed from the code operation (unless they are completely disconnected from the rest of the code) and functioning auxiliary qubits and connections are only removed if they are not needed to implement the (minimally split) n-gons formed from the reducing and splitting process. Moreover, it minimizes the damage to the code in terms of the gaps created in the code, i.e. the number and size of superplaquette operators which effectively reduce the code and fault distances. Since the above strategy is more efficient with respect to removing components and minimizes the damage to the code, we expect it to improve performance with respect to using the code reduction strategy of Ref. [17].

The strategy presented here applies rather generally to different realizations of the surface code (and possibly other codes), assuming they have certain reasonable properties. One such assumption is that there are natural circuits for measuring all the possible reduced and split plaquette stabilizers that only use previously existing components. The application of step 1 only depends on the code, not the detailed realization. In contrast, the application of steps 2 and 3 will depend on the details of the particular realization.

We now consider the implementation of this strategy for our measurement-based realization of the surface code in detail. The n-gon stabilizer measurement circuits from Sec. 4 provide all of the circuits that we need in order to implement our dead components strategy. These n-gon stabilizer

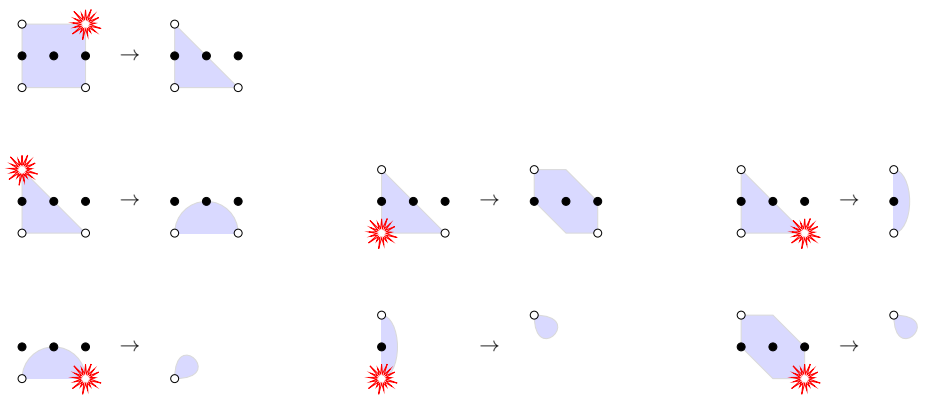


Figure 10: The possible reductions of Z-type n-gons for step 1, where dead data qubits (denoted by starbursts) are removed from the code operation. (Reductions related to these by rotations and reflections are not shown separately.) The reductions for X-type n-gons may be obtained from these by 90 degree rotations. Even though this step only considers dead data qubits, the auxiliary qubits are displayed to show the collateral removal of live auxiliary qubits that may occur when removing dead data qubits. Ignoring the auxiliary qubits, the same reduction of n-gons can be used for any realization of the surface code. The full reduction of each n-gon in this step can be determined through an iterative process where dead data qubits are removed one at a time until no dead ones remain, i.e working down through this figure.

circuits share some nice features when used for reducing and splitting n-gons. The first is that none of them introduce new physical measurements to the set of measurements needed to operate the code; a given measurement is either removed from the circuit or reduced from a pairwise to a single-qubit measurement of the same type. (We note that all single-qubit Pauli measurements are required for operation, even though they may not all occur in the 4-gon circuits.) Thus, it is a relatively simple matter to change the operation of a plaquette in a manner that reduces or splits the plaquette into smaller n-gons. Additionally, all of these n-gon circuits interlock, so we can operate all plaquettes of the same type on the same schedule, without further modification.

We can now consider each of the three steps for determining the modification of the code in the presence of dead components. The modifications for step 1, where plaquettes are modified by removing dead data qubits, are shown in Fig. 10. The modifications for step 2, where plaquettes are modified by removing dead auxiliary qubits, are shown in Fig. 11. The modifications for step 3, where plaquettes are modified by removing dead connections (pairwise measurements), are shown in Fig. 12. In each of these figures, we display the different possible scenarios (up to rotations and reflections) for removing a single dead component at a time. In order to remove all dead components from operation, we check components of a given step one at a time and apply the corresponding reduction or splitting shown in the figures, working down through possible scenarios, and repeat for the next component until all dead components are removed.

As a demonstration of the generality of our proposed strategy for surface code realizations, we can apply it to the measurement-based pentagonal tiling realization of Ref. [9] and the CNOT gate-based realization. We show the details for these in Appendix C

In order to demonstrate the advantages of our proposed strategy and, in particular, how steps 2 and 3 make our protocol differ from that of Ref. [17], it is useful to consider some example scenarios

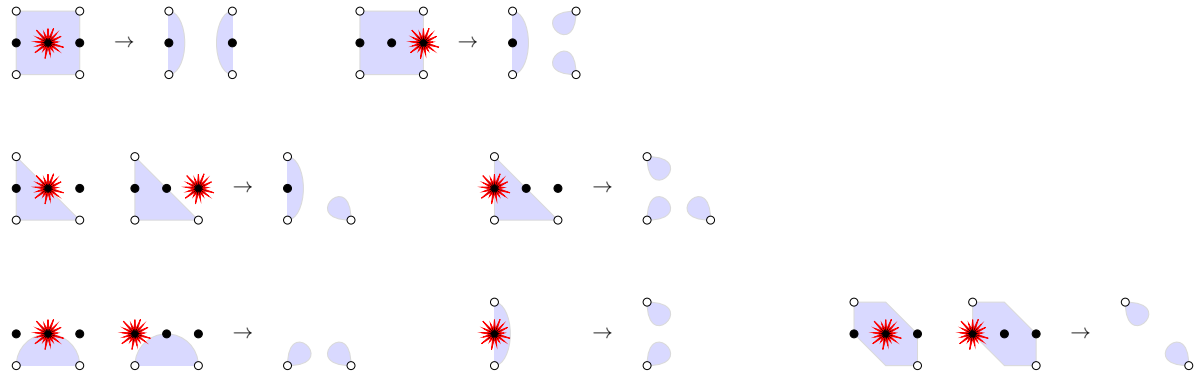


Figure 11: The possible splittings of Z-type n-gons for step 2, where dead auxiliary qubits (denoted by starbursts) are removed from the code operation. (Splittings related to these by rotations and reflections are not shown separately.) The splittings for X-type n-gons may be obtained from these by 90 degree rotations. In some cases, live auxiliary qubits will be collaterally removed as a result of removing dead auxiliary qubits. The full splitting of each n-gon in this step can be determined through an iterative process where dead auxiliary qubits are removed one at a time (together with any collateral loss), until no dead ones remain, i.e working down through this figure.

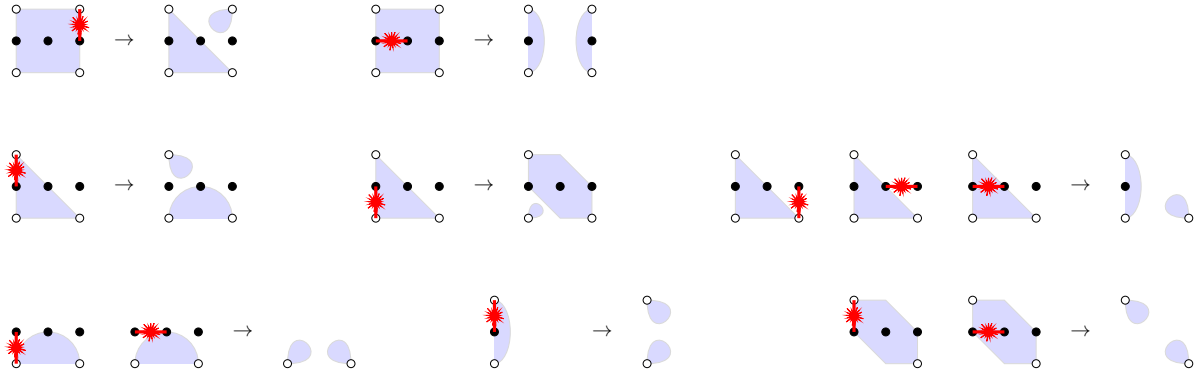


Figure 12: The possible splittings of Z-type n-gons for step 3, where dead connections (denoted by starbursts) are removed from the code operation. (Splittings related to these by rotations and reflections are not shown separately.) The splittings for X-type n-gons may be obtained from these by 90 degree rotations. In some cases, live auxiliary qubits will be collaterally removed as a result of removing dead connections. The full splitting of each n-gon in this step can be determined through an iterative process where dead connections are removed one at a time (together with any collateral loss), until no dead ones remain, i.e working down through this figure.

of dead components in detail. We show the modifications of plaquettes for several dead component scenarios, as well as the Z- and X-measurement rounds' corresponding measured stabilizers (damaged and undamaged) and the superplaquette stabilizers that survive from round to round. (Recall that when there are no dead components, all plaquette stabilizers survive from round to round.)

In Fig. 13, we show the modification of the plaquette stabilizers associated with one dead data qubit. Following the protocol of Ref. [17], this modification would also be used in the case where that data qubit was live, but any of the four connections involving that qubit were dead. With

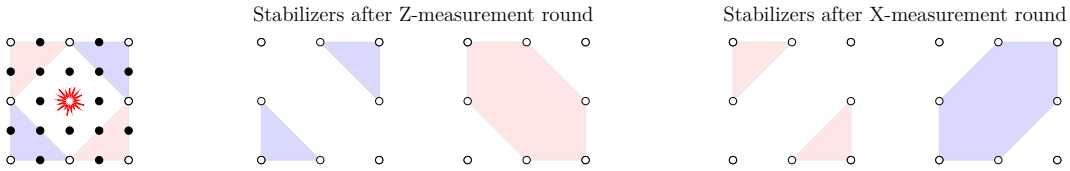


Figure 13: Configuration for one dead data qubit.

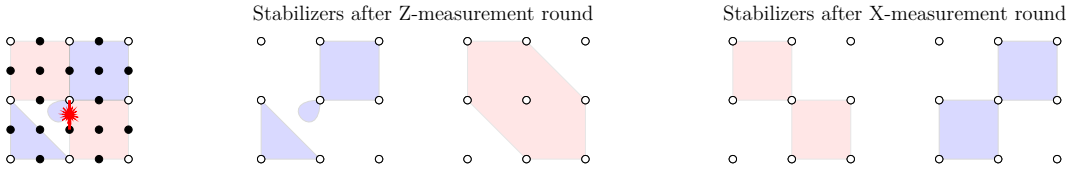


Figure 14: Configuration for one dead connection to a data qubit.

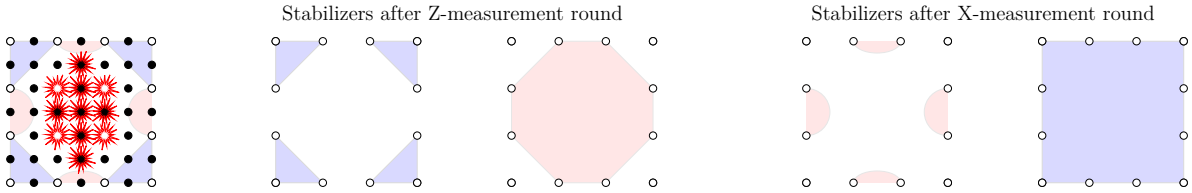


Figure 15: Configuration for a dead plaquette.

this modification, the code distance is reduced by one. In Fig. 14, we show our modification of the plaquette stabilizers associated with one dead connection between a data qubit and an auxiliary qubit. Comparing to the modification in Fig. 13, we see that the stabilizers only differ from the undamaged stabilizers in the Z-measurement round and that the code distance is reduced by one for only the Z logical string operators.

In Fig. 15, we show the modification of the plaquette stabilizers associated with one dead plaquette, i.e. four dead data qubits. Following the protocol of Ref. [17], this modification would also be used in the case where the data qubits were live, but any of the auxiliary qubits of the plaquette were dead, or each of those data qubits had a dead connection. With this modification, the code distance is reduced by two. In Fig. 16, we show our modification of the plaquette stabilizers associated with a plaquette for which the A and C auxiliary qubits are dead, or the four connections between the data qubits and the auxiliary qubits of that plaquette are dead. We again see that the stabilizers only differ from the undamaged stabilizers in the Z-measurement round and that the code distance is reduced by two for only the Z logical string operators. In Fig. 17, we show our modification of the plaquette stabilizers associated with a plaquette for which only the B auxiliary qubit is dead, or for which either of the connections involving the B qubit is dead. Again, the stabilizers only differ from the undamaged stabilizers in the Z-measurement round and, in this case, the distance is reduced by two only for Z logical string operators in the vertical direction. This may not affect the code distance if the logical qubit is encoded such that the logical Z string operators are aligned in the horizontal direction.

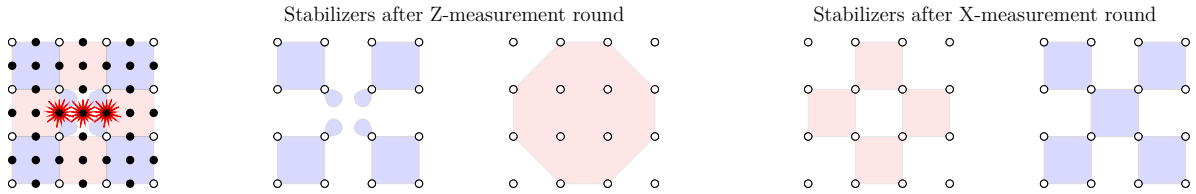


Figure 16: Configuration for a plaquette with dead auxiliary qubits A and C, or dead connections between data qubits and auxiliary qubits A and C.

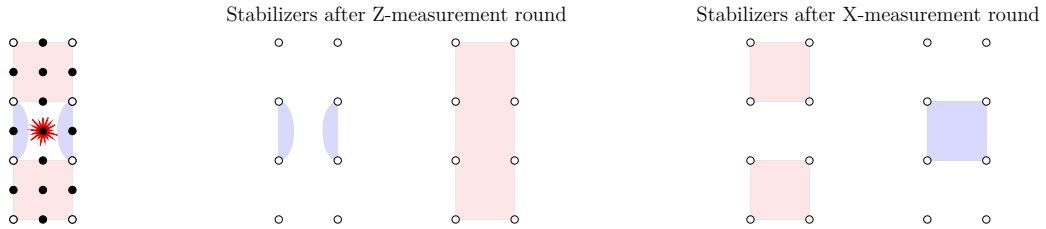


Figure 17: Configuration for a plaquette with dead auxiliary qubit B or a dead connection to this qubit.

6 Implementation in Majorana Hardware

We now consider the implementation of our measurement-based realization of the surface code in Majorana hardware [6]. As this was the original motivation for generating our new surface code realization, it should not be surprising that the layout and measurements involved are very simple for this hardware. We consider rectangular arrays of Majorana tetron qubits, which are formed from two Majorana wires connected by a superconducting spine. Each tetron possesses four Majorana zero modes (MZMs), one at each endpoint of each Majorana wire. A pair of MZMs combine into a fermionic mode, i.e. the joint fermionic parity of two MZMs corresponds to a two-level system. However, since each tetron is a floating superconducting island with a charging energy, there is an overall parity constraint of the four MZMs of a tetron. In this way, the tetrons form a qubit, where the joint parity operators associated with different MZM pairs correspond to the different Pauli operators of the qubit. In particular, when the jth MZM of a tetron has corresponding Majorana operator γ_j, we use the convention where the joint fermionic parity operator $P_{jk} = i\gamma_j\gamma_k$ of MZMs j and k map to Pauli operators according to

$$X = P_{23} = P_{14}, \tag{6}$$

$$Y = P_{13} = -P_{24}, \tag{7}$$

$$Z = P_{12} = P_{34}. \tag{8}$$

See Ref. [6] for more details.

In order to facilitate measurements of these operators, one needs additional components around the tetrons that can couple to MZMs as desired to form interference loops that enable measurement of the joint parity of all the MZMs in the interference loop. (Such measurements always involve exactly two MZMs from each tetron qubit involved in the measurement.) The components utilized for this include semiconductor rails running along the short direction of the tetrons (between columns of tetrons in a rectangular array), which enable (electrostatic) gate-defined quantum dots

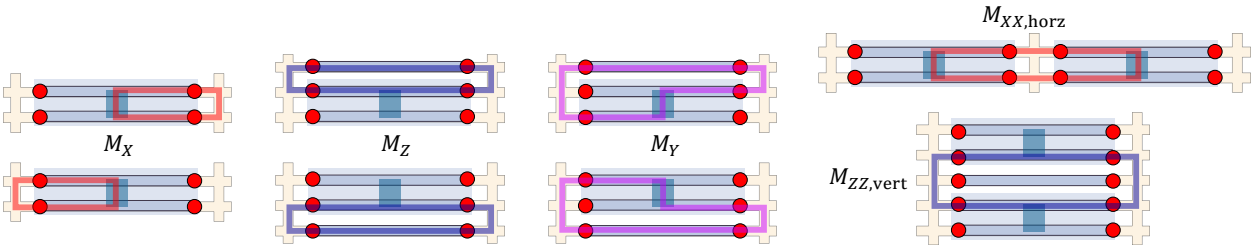


Figure 18: Measurement loops in Majorana hardware corresponding to the measurements needed for our code. MZMs (red circles) are located at the end points of topological wires (gray lines). Two topological wires connected by a trivial superconducting spine (dark gray) form a tetron. Tetrons can be connected to each other or coherent links through semiconductor segments (tan).

with gate-controlled couplings to the MZMs. There are also "coherent links," which are single floating Majorana wires (with a MZM on each of its two endpoints), which are located between tetrons running along the long direction. These facilitate the creation of interference loops involving MZMs on the opposite sides of tetrons in the long direction. We display schematic drawings of this hardware and some of the measurement loops in Fig. 18. The measurement loops shown serve to define our basis conventions with respect to the hardware, and the same choice is used for all tetrons. Once the interference loops are turned on, the joint parity of all the MZMs included in the loop can be measured, for example by probing the quantum capacitance of the coupled tetron-quantum dot system (which will exhibit parity dependence) using microwave resonators.

We expect that the fidelities of such MZM parity measurements will generally decrease with increasing length of semiconductor, number of MZMs, number of coherent links, and number tetrons utilized in the corresponding measurement loops [7], that is

$$f_{M_X} > f_{M_Z} > f_{M_Y} > f_{M_{XX:\mathrm{horz}}} > f_{M_{ZZ:\mathrm{vert}}} > f_{M_{YZ:\mathrm{vert}}} > f_{M_{YY:\mathrm{vert}}} > \cdots . \tag{9}$$

Here, we have indicated the horizontal or vertical direction on pairwise measurements, because it makes a significant difference in the difficulty of the measurement, e.g. $M_{ZZ:\mathrm{horz}}$ would require the additional use of two coherent links as compared with $M_{ZZ:\mathrm{vert}}$. This provides a rough guide for optimizing codes or computations with respect to the measurements utilized.

We now examine the implementation of our measurement-based realization of the surface code in a rectangular array of tetrons. We notice that our code only uses M_X, M_Z, $M_{XX:\mathrm{horz}}$, and $M_{ZZ:\mathrm{vert}}$ measurements (for logical memory). These are the two simplest single-qubit measurements and two simplest two-qubit measurements for this hardware. Thus, our measurement-based realization of the surface code is highly optimized with respect to the set of measurements in Majorana hardware.

One significant aspect of device design for Majorana hardware is whether a single rail or double rail of semiconductor is used between the columns of tetrons. The semiconductor rails are where most of the operational activity is concentrated in this hardware, e.g. the quantum dots, coupling to MZMs, and measurements. As such, utilizing double-rail semiconductors constitutes an increase in fabrication and control difficulties, which may also translate into higher error rates. However, the positive aspect of the trade-off is that using double-rails allows independent measurements on adjacent tetron columns to be performed without conflict. For single-rail layouts, the interference loops for certain configurations of measurements would overlap and hence could not be performed

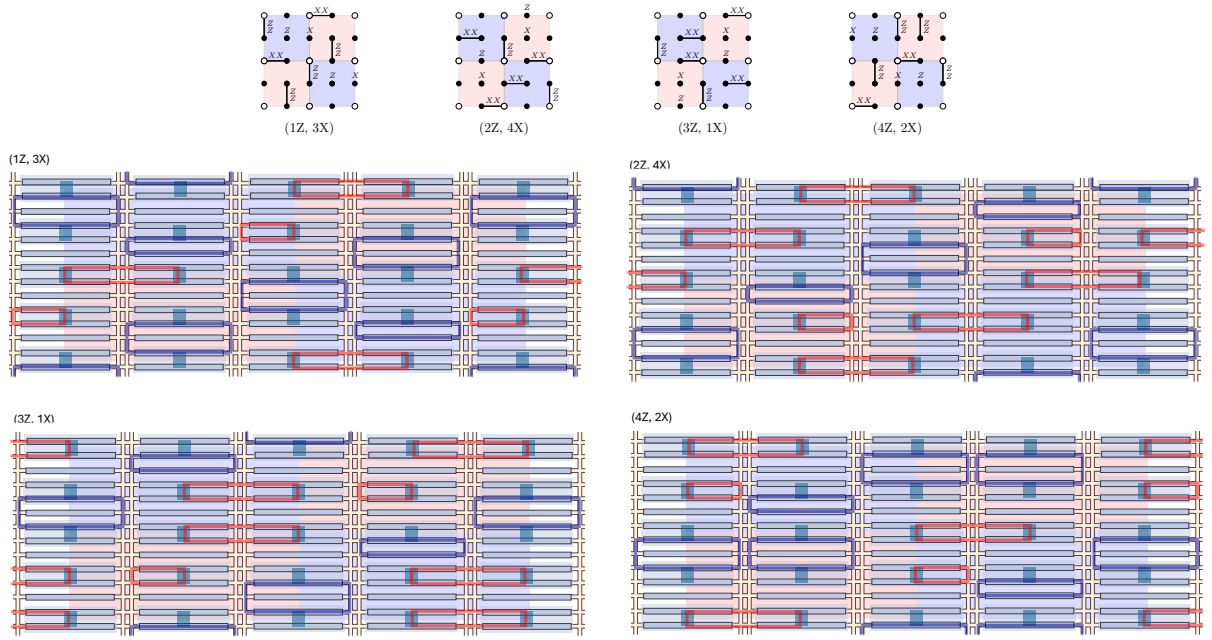


Figure 19: Interference loops of measurements for the implementation of our measurement-based surface code realization in Majorana hardware with double-rail semiconductor layouts. This layout avoids loop conflicts, allowing for the (minimal) period 4 operation schedule.

at the same time. (For double-rail layouts, there may, in practice, be unwanted cross-talk between such measurements when performed simultaneously, since their interference loops would necessarily be in close proximity to each other.) With this in mind, it is useful to consider the implementation of codes in both single-rail and double-rail layouts to appreciate how this difference affects performance in Majorana hardware.

We can implement our code in Majorana hardware with the operation schedule of Eq. (2) using rectangular arrays with double-rail semiconductors, as shown in Fig. 19. Examining the interference loops in the steps, we see that attempting to implement this operation schedule on an array with single-rail semiconductors would result in loop conflicts in steps $(1Z, 3X)$ and $(4Z, 2X)$. A naïve resolution of this would be to split each of the steps with conflicts into two steps, distributing the measurements between them so that no step contains a conflict. However, a more efficient resolution is possible, which we show in Fig. 20. In particular, we split step $(4Z, 2X)$ into two steps, $(4Z, -)$ and $(-, 2X)$, and shift one of the conflicting measurements from step $(1Z, 3X)$ into the second of these split steps. Denoting step $1Z$ without the M_{Z_B} measurement as $1Z'$ and the M_{Z_B} measurement as $1Z''$, the resulting pipelining for single-rail layouts is

$$\cdots, (1Z', 3X), (2Z, 4X), (3Z, 1X), (4Z, -), (1Z'', 2X), \cdots . \tag{10}$$

This operation schedule for single-rail semiconductor layouts has a five step period, which is a relatively mild slow down. (We note that there are other possibilities for redistributing the measurements among a five step cycle that will be compatible with single-rail layouts.)

For the hook-preventing circuits described in Sec. 3, we can implement the code using any of the

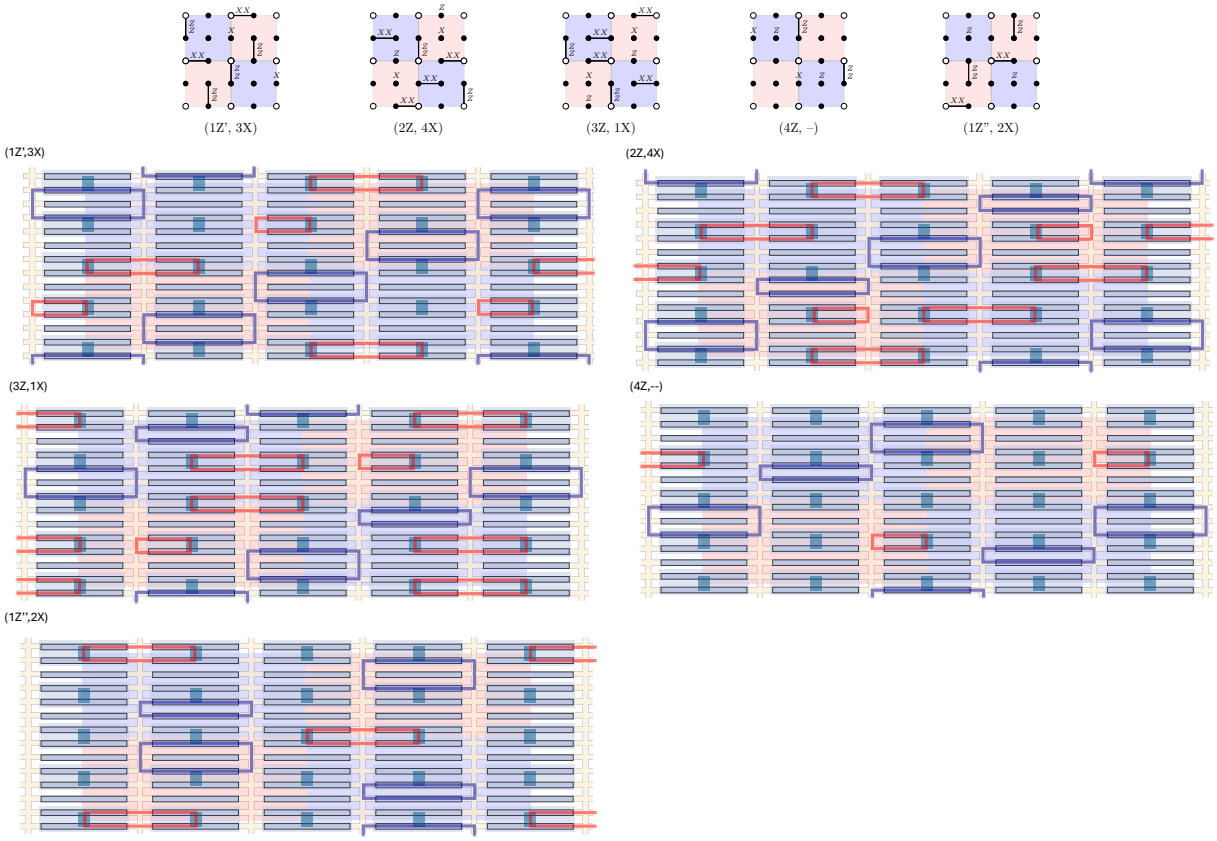


Figure 20: The implementation of our measurement-based surface code realization in Majorana hardware with single-rail semiconductor layouts requires resolution of loop conflicts in two steps. A resolution with period 5 operation schedule is shown here.

operation schedules shown in Fig. 7 with double-rail layouts. For single-rail semiconductor layouts, we can resolve interference loop conflicts while only increasing the operation period by one step (from seven steps to eight steps), e.g. by modifying the pipelining option 2 given in Eq. (5) in a similar manner as described above.

It is worth comparing the implementation of our surface code realization in Majorana hardware to that of the Floquet code on the 4.8.8 lattice, as considered in Ref. [5]. The Floquet code can also be implemented on a rectangular array of qubits, where it utilizes measurements that we denote as $M_{XX:\mathrm{horz}}$, $M_{ZZ:\mathrm{vert}}$, and $M_{YY:\mathrm{vert}}$ (in equal proportions). For double-rail semiconductor layouts, the 4.8.8 Floquet code could operate on a six step period, in which no qubits are idle. However, in order to implement this code on single-rail semiconductor layouts, it appears that interference loop conflicts can only be resolved by splitting each step into two steps, doubling the operation period and introducing the possibility of faults on idle qubits for half the qubits at each step. As such, the comparative performance of our code vs. 4.8.8 Hastings-Haah is likely to improve when the more realistic hardware conditions are taken into consideration.

7 Performance

In this section, we assess the performance of the codes presented in Secs. 2, 3, and 6, using the circuit noise model presented in Ref. [8]. In this noise model, each measurement fails independently with probability p_{physical}. When a measurement fails, it acts as an ideal measurement followed by an error drawn uniformly from the set of nontrivial errors supported on the qubits involved in the measurement. For a single-qubit measurement, the error is drawn from the set $\{(P_1, F)\} - \{(I, 0)\}$. For a two-qubit measurement fault, the error is drawn from the set $\{(P_1 \otimes P_2, F)\} - \{(I \otimes I, 0)\}$. Here, $P_1, P_2 \in \{I, X, Y, Z\}$ are Pauli errors acting on the support of the measurement and $F \in \{0, 1\}$ corresponds to a readout error, i.e. a bit flip of the measurement outcome.

Some of the code variants we consider include steps in which qubits are idle. The noise model also includes faults on idle qubits, with errors drawn from the set $\{X, Y, Z\}$. At each time step that a qubit idles, we assign the same error rate p_{physical} for an idling error to occur. However, since this is likely to overestimate the relative error rate of idling compared to measurements for our hardware of interest, it is useful to compare code performance with and without faults on idle qubits. We will specify when our analysis does not include faults on idle qubits. The distinction will be significant for the hook-preventing circuit and when analyzing implementations in Majorana hardware with single-rail semiconductor layouts.

7.1 Decoding graph construction

We first provide an overview of how we construct a decoding graph, on which we use PyMatching v2 to efficiently perform minimum weight perfect matching via the "sparse blossom" algorithm [22]. We use the spacetime circuit formalism developed in Ref. [23], and the splitting decoder of Ref. [24]. We start with a Clifford circuit consisting of single- and two-qubit measurements. In the absence of errors, the measurements within this circuit satisfy nontrivial correlations, called detectors. More formally, a detector consists of a set of measurements for which the joint parity of their outcomes is fixed in the absence of errors [19]. Let $\{m_\alpha\}$ be the set of measurement outcomes for the circuit, where α runs over all spacetime coordinates. The set of detectors form a classical parity check code over the measurement outcomes. We define the matrix D to be such that each row forms a check of the corresponding classical parity check code. Here, we assume $D_{j\alpha}$ and m_α take values in $\mathbb{F}_2$ with $\cdot$ the usual product and addition mod 2. Then $\sum_\alpha D_{j\alpha} \cdot m_\alpha$ takes on a fixed value in the absence of errors, and forms the jth detector. We can find all detectors using the fault propagation as described in Ref. [23] [see Algorithm 1 in this reference, and note that they refer to detectors as "checks" (as is also the case in Ref. [14])].

A spacetime error chain corresponds to a collection of Pauli errors and readout errors on the circuit. An error chain is detectable if it triggers one or more detectors. In practice, we choose a basis of detectors and use those to form a decoding graph or hypergraph. We label each detector by an integer j, and refer to the set of detectors $\{j, k, ...\}$ that are triggered by an error chain (meaning that the sum of the measurement outcomes mod 2 differs from its value in the absence of errors) as the error chain's syndrome. Our ability to perform error correction depends on the ability to distinguish between equivalence classes of spacetime error chains, based on their syndromes.

Each vertex in the decoding graph can (in our chosen basis) be associated to a Z or X plaquette, as well as a time coordinate. We note that for a repeated plaquette stabilizer measurement, using

the measurement circuits defined in Fig. 2 (original circuit) or Fig. 6 (hook-preventing circuit), there are several low-weight detectors associated with each plaquette (meaning detectors with a small number of contributing measurement outcomes), along with the high-weight detector that corresponds to the repeated measurement of the surface code stabilizer associated to the plaquette. The low-weight detectors are formed by repeated single auxiliary qubit measurements—such as the repeated M_{X_B}—or (in the hook-preventing circuit) by the alternating repeated pairwise auxiliary qubit measurements.

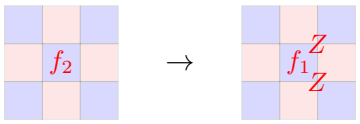
Given a basis of detectors together with a set of generative fault configurations $\mathcal{F} = \{f_1, f_2, \cdots\}$, which we take to be degree one faults in the circuit noise model, a decoding hypergraph $H = V, E$ is defined as follows. Each vertex $v_j \in V$ corresponds to a detector j. A set of vertices $\{v_j, v_k, ...\}$ are connected by a hyperedge $e \in E$ if there exists a fault $f \in \mathcal{F}$ with the corresponding syndrome $\{j, k, ...\}$. The weight of the hyperedge is defined from the probability of the fault occurring. This construction generally does not result in a decoding *graph*, as certain faults in $\mathcal{F}$ may trigger three or more detectors and give rise to hyperedges. While matching on a graph can be performed in polynomial time, matching on a hypergraph is generally an NP-hard problem [25]. As described in Ref. [24], it is possible for some decoding hypergraphs to split hyperedges into edges in a consistent way that allows for successful decoding. This is done through the construction of a *split noise model*. We define *primitive faults* as faults that either trigger one detector (1-faults), or that trigger two detectors without being decomposable into 1-faults (2-faults). By construction, a set of generative fault configurations containing only primitive faults will only give rise to edges. To define the split noise model, any non-primitive fault in $\mathcal{F}$ is decomposed into primitive faults, such that they together trigger the same set of detectors as the original non-primitive fault. Each of these new faults is assigned the same error rate as the original n-fault, and is added to a new set of generative fault configurations, $\tilde{\mathcal{F}}$, together with the primitive faults in $\mathcal{F}$. $\tilde{\mathcal{F}}$ approximates $\mathcal{F}$ while containing only primitive faults, and is used to define a decoding graph on which minimum weight matching is performed. The matching decoder that uses this graph is referred to as a splitting decoder.

7.2 Dynamic re-weighting of the decoder graph edges

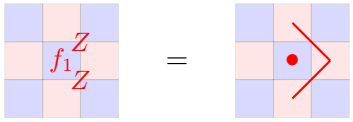
While the splitting decoder works straightforwardly for the original circuit defined in Fig. 2, the hook-preventing circuit defined in Fig. 6 contains additional detectors, which complicate the use of a splitting decoder. These increase the weight of the syndromes for some circuit noise errors in $\mathcal{F}$, which in turn affects the performance of the splitting decoder such that the fault distance is effectively halved. To avoid this issue, we dynamically change the weights of the edges in the decoding graph in the presence of certain syndromes, as described below. This dynamic re-weighting is inspired by Ref. [26], where soft information is used to dynamically determine edge weights (rather using than syndromes, as in the present context). With the dynamic re-weighting added for the hook-preventing circuit, we find that the splitting decoder is successful in all cases considered here.

We illustrate the need for re-weighting with an example. We recall the measurement sequence in the hook-preventing circuit for a Z-plaquette, shown in Fig. 6. The repeated pairwise auxiliary measurements $M_{X_A X_B}$ and $M_{X_B X_C}$ in steps 2-5 give rise to two low-weight detectors. Using superscripts to indicate the time steps of measurements, the first low-weight detector is given by the sum of the measurement outcomes from $M^{2Z}_{X_A X_B}$ and $M^{4Z}_{X_A X_B}$, and the second by the sum of the measurement outcomes from $M^{3Z}_{X_B X_C}$ and $M^{5Z}_{X_B X_C}$. In our chosen basis of detectors, a single

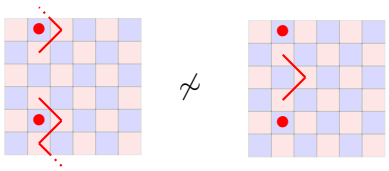
readout error on the second repetition of the measurements in these pairs, e.g. $M^{4Z}_{X_A X_B}$, will trigger not only the low-weight detector, but also two high-weight detectors associated to neighboring X-plaquettes. As such, a fault that results in such a readout error is a non-primitive fault. It is decomposed onto a 1-fault resulting in a readout error on $M^{2Z}_{X_A X_B}$, which in the chosen basis triggers only the low-weight detector, and two 2-faults that together result in Z-errors on two data qubits.[5] The spatial distribution of the errors is indicated in the following figure, where we denote the faults that result in readout errors on the first and second $M_{X_A X_B}$ by f_1 and f_2:



The primitive Z 2-faults above correspond to edges between the decoding graph vertices that represent the high-weight detectors associated to the neighboring X-plaquettes. The primitive f_1 1-fault corresponds to a "dangling edge," in the graph as it only connects to one single vertex, i.e. the low-weight detector. We represent such dangling edges as dots in the pictures below. In terms of edges, the three primitive faults are represented as



In the presence of multiple non-primitive faults of this type, the matching can fail. Consider the following edges representing two f_2 faults on a 6 by 6 torus (left), and another, logically inequivalent edge configuration (right) with the same syndrome:



As non-primitive f_2 faults appear on *all* plaquettes in the set of generative fault configurations $\mathcal{F}$, all non-dangling edges in the above picture by symmetry come with the same weight. Therefore, the minimum weight matching prioritizes the path with fewer edges, and the decoding induces a logical error. To get around this mismatch, we temporarily assign weight zero to all non-dangling edges that correspond a split f_2 fault (meaning that they come for free) whenever the dangling edge is "lit up" by a fault configuration, i.e. whenever the corresponding low-weight detector is triggered. With this temporary re-weighting, the matching will not penalize the longer path and the decoding succeeds.

Other faults need to be treated in the same fashion. When constructing the decoding graph for the hook-preventing circuits, we first identify the faults whose splitting will require dynamical re-weighting. These correspond to faults of the form $(I, 1)$, i.e. pure readout errors, that light up three detectors. These faults are split into three primitive faults. One of these primitive faults in

[5] These can be reversed by an equally natural basis choice, so that a readout error in the first repetition triggers three detectors, and a readout error in the second repetition only triggers one detector.

the splitting is a 1-fault that triggers only a single detector v_j. The other two primitive faults in the splitting are 2-faults, and during decoding the corresponding two edges in the decoding graph are assigned weight zero whenever j is present in the observed syndrome.

7.3 Results

In this section, we analyze the performance of our code and compare it directly with that of the 4.8.8 Floquet code, which represents the state of the art for pairwise measurement-based codes optimized with respect to Majorana hardware. Files providing computer parsable descriptions of the circuits used in our simulations are provided in the supplementary material. We estimate the logical failure rates by running Monte Carlo simulations with up to 10^8 trials for a series of increasing code sizes and a fixed set of physical error rates $10^{-4} \leq p_{\text{physical}} \leq 1.5 \times 10^{-2}$. At sufficiently large code size and low p_{physical}, we observe no failures in the 10^8 trials and, thus, do not include the corresponding point in the plots. The shaded regions in performance plots [Figs. 21, 23, 25, and 26] represent 95% credible intervals of a posterior beta distribution given the observed number of logical failures and completed trials, assuming a uniform prior distribution for the logical error rate; the points represent the median of this posterior distribution. In this paper, we extract threshold values by finding the intersection of the two largest simulated code sizes via a linear interpolation *in log-log space* of the obtained $(p_{\text{physical}}, p_{\text{logical}})$ data. We ignore sampling uncertainties in these threshold estimates, since sampling error is generally negligible (e.g., occurring in the 4th significant figure in the p_{logical} estimates) at the relatively high error rates encountered near threshold. There are, however, systematic uncertainties due to finite code size and our employed interpolation procedure. We did not attempt to estimate error bars for the threshold values, since the difference of noise model used from a realistic noise model undoubtedly results in more significant deviations. Note that we expect the finite size effects to cause *underestimation* of the threshold. For a more rigorous threshold estimation procedure, see, e.g., Ref. [14]. We extract "pseudo-thresholds" for a particular system size by locating the error rate where $p_{\text{logical}} = p_{\text{physical}}$, again obtained via linear interpolation in log-log space.

We first compare the performance for our original circuits and pipelining (as described in Sec. 2) on a rotated surface code patch using boundary conditions that make the hook errors benign (as explained in Sec. 4) with the performance of the 4.8.8 Floquet code on a planar patch with rectangular boundary conditions (as in Ref. [5]). For implementation in Majorana hardware, these both correspond to realizations that would require double-rail semiconductor layouts. The performance results for these two cases are shown in Fig. 21. Notably, we find a fault-tolerance threshold of 0.66% for our code and 1.3% for the 4.8.8 Floquet code.[6] For the smallest code size ($d_{\text{f}} = 3$), the pseudo-thresholds are comparable for the two codes at approximately 0.096% for the surface code and 0.16% for 4.8.8 Floquet code.

When evaluating code performance, it is helpful to go beyond the threshold and consider the resource requirements for obtaining a given target logical error rate $p_{\text{logical}}^{\text{target}}$ for the simulated memory experiment. In Fig. 22, we plot the qubit count, circuit depth, and spacetime footprint (i.e.,

[6]The improvement of the threshold value for 4.8.8 Floquet code compared to the previous simulations of Ref. [5] are likely due to some combination of (a) our use of an improved decoder, (b) larger considered code sizes, and (c) improved sampling statistics.

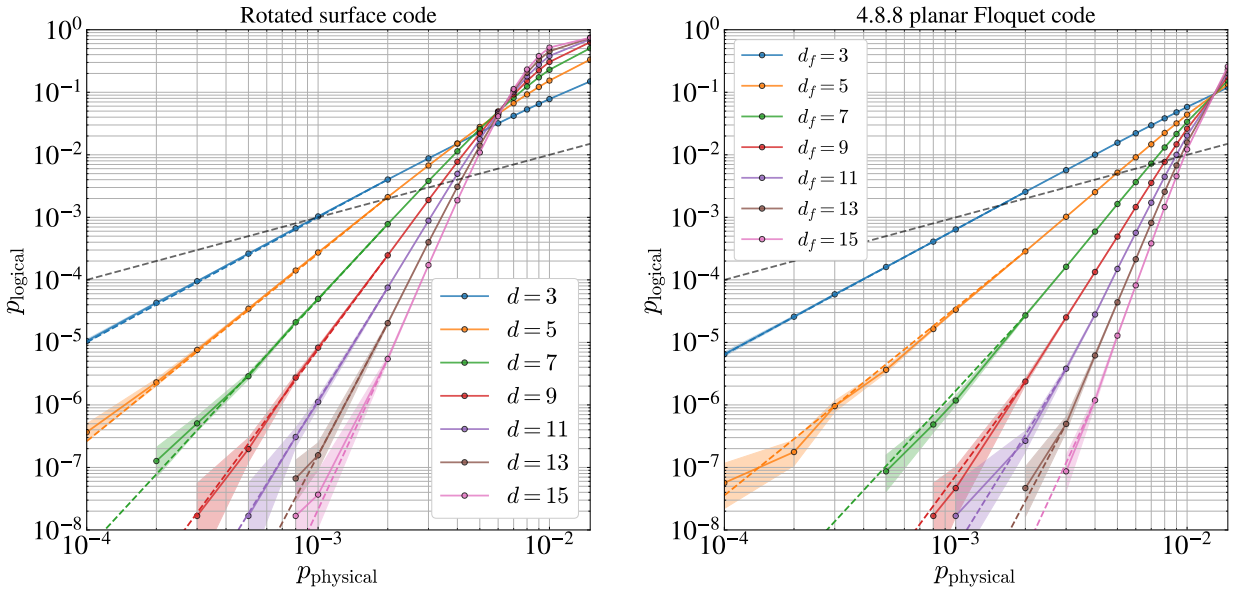


Figure 21: (left) Performance results for our pairwise measurement-based surface code realization described in Sec. 2 on a rotated surface code patch using boundary conditions that make the hook errors benign (see Sec. 4). (right) Performance results for the 4.8.8 Floquet code on a planar patch with rectangular boundary conditions (as in Ref. [5]). For implementation in Majorana hardware, these require double-rail semiconductor layouts. Fault-tolerance thresholds are found to be 0.66% for our code and 1.3% for the 4.8.8 Floquet code.

qubit count multiplied by circuit depth) required to achieve logical error rates of $p_{\text{logical}}^{\text{target}} = 10^{-8}$, 10^{-12}, and 10^{-15} per d_{f} rounds of syndrome measurement, for physical error rates in the range $10^{-6} \leq p_{\text{physical}} \leq 10^{-3}$. In Table 2, we list the corresponding fault distances d_{f} required to reach these target logical error rates for physical error rates $p_{\text{physical}} = 10^{-6}$, 10^{-5}, 10^{-4}, and 10^{-3}. Within the context of this simplified error model, we see that our surface code is competitive in terms of these resource requirements, especially at error rates $p_{\text{physical}} \lesssim 10^{-4}$, and it has substantially narrowed the gap from the original requirements of the pairwise measurement-based surface codes in Ref. [8] at higher error rates $p_{\text{physical}} \approx 10^{-3}$, where the Floquet code previously enjoyed a nearly two full orders of magnitude reduction in spacetime footprint at $p_{\text{logical}}^{\text{target}} = 10^{-12}$ [see Fig. 9 of Ref. [5]].

To obtain these resource estimates, we have taken the following approach. For each code and code size, we expect the low p_{physical} behavior to be dominated by circuit noise faults of weight $(d_{\text{f}} + 1)/2$ [27]. Rather than assuming that this form persists all the way to p_{physical} on the order of the threshold, we select, for each code size, a characteristic reference point $(p_{\text{physical}}^{\text{ref}}, p_{\text{logical}}^{\text{ref}})$ in the sub-threshold regime of the empirical data and assume that for $p_{\text{physical}} \leq p_{\text{physical}}^{\text{ref}}$, the logical error rate is governed by the form

$$p_{\text{logical}} = p_{\text{logical}}^{\text{ref}} \left(\frac{p_{\text{physical}}}{p_{\text{physical}}^{\text{ref}}} \right)^{(d_{\text{f}}+1)/2} . \tag{11}$$

The colored dashed lines in the performance plots represent these chosen sub-threshold estimates

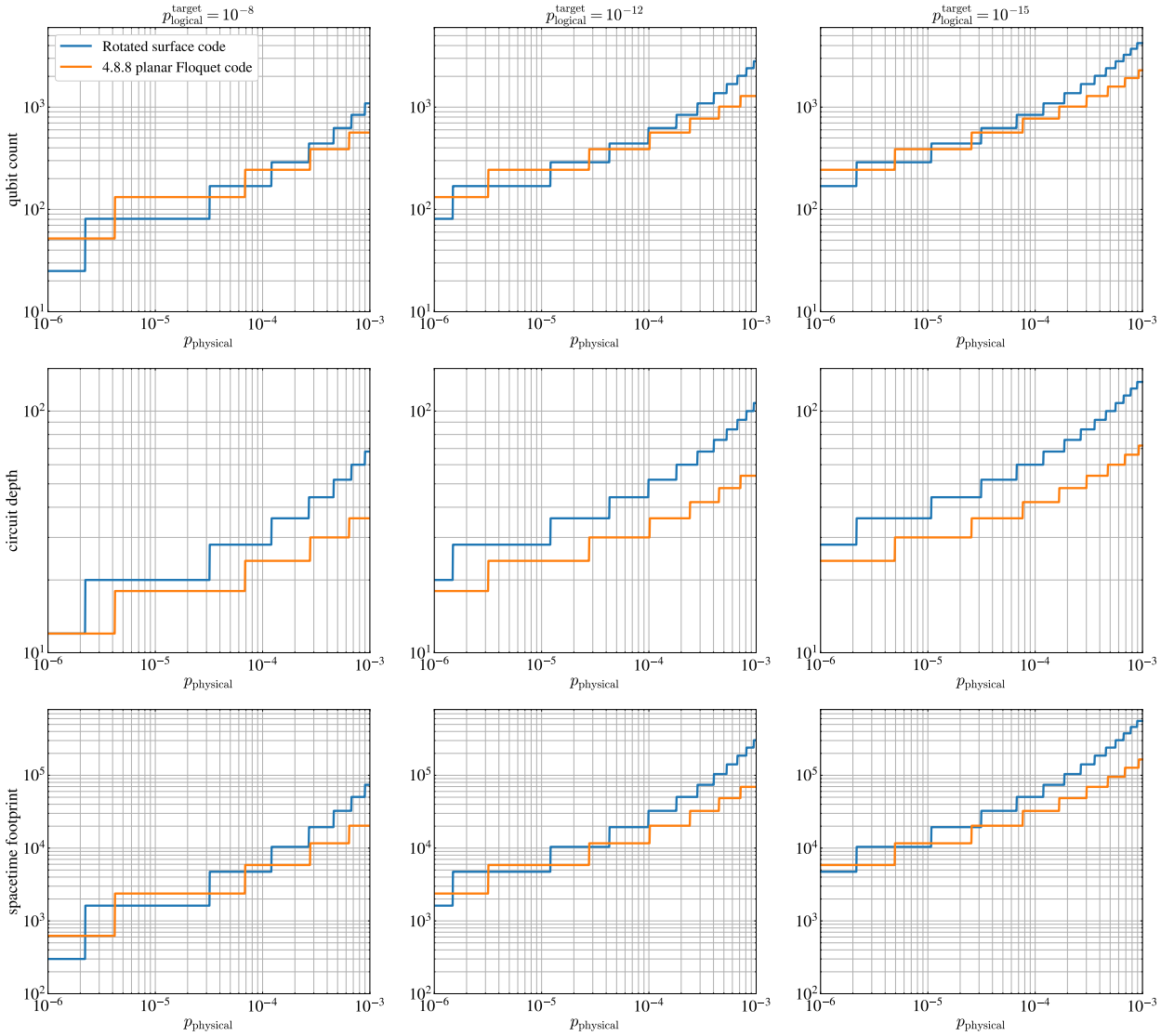


Figure 22: Resource requirements to reach target logical error rates of $p_{\text{logical}}^{\text{target}} = 10^{-8}$, 10^{-12}, and 10^{-15} per d_{f} rounds, comparing our realization of the surface code and the 4.8.8 planar Floquet code. The respective target logical error rates correspond to columns from left to right, and the resource quantities being considered correspond to rows, from top to bottom: qubit count, circuit depth, and spacetime footprint (i.e., qubit count times circuit depth).

	$p_{\text{logical}}^{\text{target}} = 10^{-8}$		$p_{\text{logical}}^{\text{target}} = 10^{-12}$		$p_{\text{logical}}^{\text{target}} = 10^{-15}$	
p_{physical}	SC	4.8.8	SC	4.8.8	SC	4.8.8
10^{-6}	3	3	5	5	7	7
10^{-5}	5	5	7	7	9	9
10^{-4}	7	7	13	9	15	13
10^{-3}	17	11	27	17	33	23

Table 2: Fault distance d_{f} required to reach a target logical error rate $p_{\text{logical}}^{\text{target}}$ of 10^{-8}, 10^{-12} and 10^{-15}, respectively, comparing our realization of the surface code and the 4.8.8 planar Floquet code. The determination of the required d_{f} is the same as in Fig. 22. (The total number of qubits is $N = O(4d_{\text{f}}^2)$ for both of these codes.)

of the logical error rate.[7] To estimate the fault distance required to hit a prescribed $p_{\text{logical}}^{\text{target}}$ at a given p_{physical}, we first fit the values of these obtained scaling forms for the sub-threshold logical error rate for all simulated code sizes $d_{\text{f}} > 3$ to the exponential form

$$p_{\text{logical}}(p_{\text{physical}}, d_{\text{f}}) = \alpha(p_{\text{physical}})e^{-\beta(p_{\text{physical}})d_{\text{f}}}. \tag{12}$$

The fits thereby obtained are of very high quality in the considered window $10^{-6} \leq p_{\text{physical}} \leq 10^{-3}$, justifying the approach *a posteriori*. Finally, we determine the smallest (odd) $d_{\text{f}} = d_{\text{f}}^{\text{target}}$ necessary for the fitted exponential form to predict $p_{\text{logical}} \leq p_{\text{logical}}^{\text{target}}$. The actual footprints can then be read off from the circuit corresponding to the required fault distance d_{f} as follows. The physical qubit count for our code on a rotated surface code patch with hook-benign boundary conditions (i.e. $d_{\text{f}} = d$) is $N = 4d^2 - 4d + 1$ and the circuit depth is counted as $4d$. For the 4.8.8 Floquet code on a patch with rectangular boundary conditions [5], the qubit count is $N = 4d_{\text{f}}^2 + 8(d_{\text{f}} - 1)$ and the circuit depth is counted as $6\lceil d_{\text{f}}/2 \rceil$.

As we are interested in implementation of these codes in Majorana hardware, where single-rail semiconductor layouts may be strongly preferable to double-rail layouts, we repeat the above analysis for single-rail variants of the two codes. For our measurement-based surface code realization, we use a modification of the circuits that is compatible with single-rail layouts, as described in Sec. 6. For the 4.8.8 Floquet code, we must split each step of the original circuits into two steps in such a way to avoid conflicts between measurement loops. There are different ways to do this, but a convenient choice is to distribute half of the measurements of each type from each step into the resulting two steps. We will not show further detail of the circuit for the single-rail variant of the 4.8.8 Floquet code, as the main point is simply that there are twice as many steps in the period and half the qubits are idle in each step. Since the single-rail variants of both of these codes have steps with idle qubits, we now repeat the above analysis for these single-rail variants. We note that the noise model we use potentially overestimates the relative error rates of idle faults as compared to measurement faults, so one can view the original results and the single-rail results as assessments

[7] In the plots, a given dashed line terminates at the chosen reference point $(p_{\text{physical}}^{\text{ref}}, p_{\text{logical}}^{\text{ref}})$. For $p_{\text{logical}}^{\text{ref}}$, we use the median of the posterior beta distribution obtained for p_{logical}.

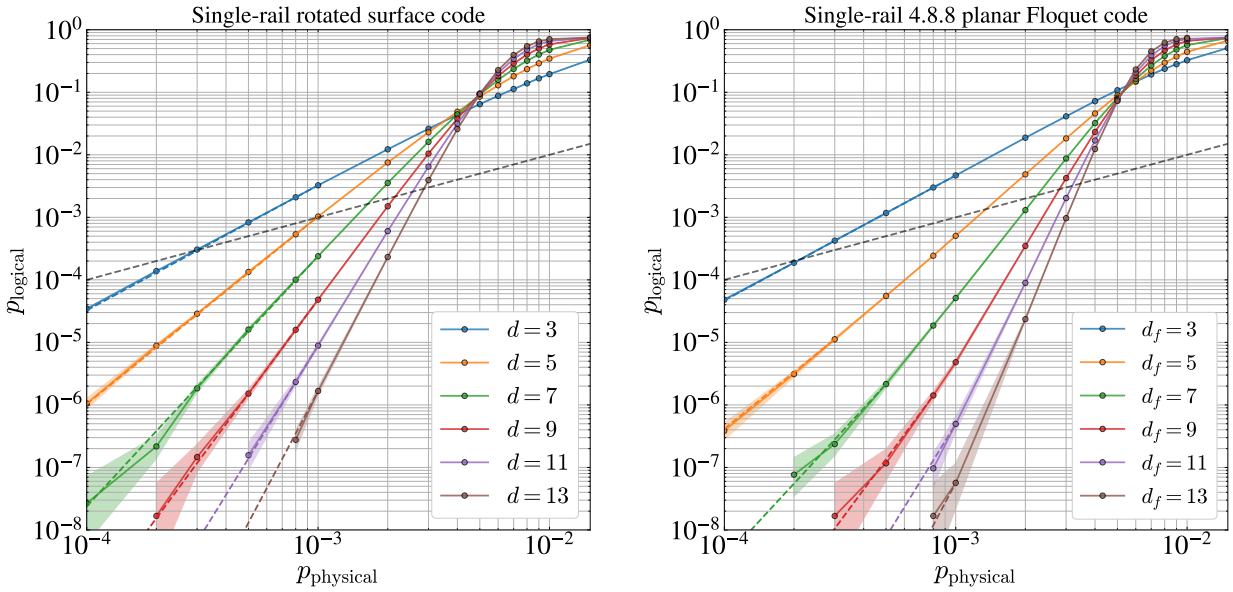


Figure 23: (left) Performance results for the single-rail variant of our pairwise measurement-based surface code realization described in Sec. 6 on a rotated surface code patch using boundary conditions that make the hook errors benign. (right) Performance results for the single-rail variant of 4.8.8 Floquet code on a planar patch with rectangular boundary conditions (see description in text). These variants can be implemented in Majorana hardware with single-rail semiconductor layouts. Fault-tolerance thresholds for these single-rail variants are found to be 0.51% for our code and 0.52% for the 4.8.8 Floquet code.

at the endpoints of a range of possible idle noise rates. (If we consider the single-rail code variants with the idle noise set to zero, we would obtain the original performance results.)

Comparing the single-rail variants of these two codes, we find that our code becomes even more competitive. From the performance results in Fig. 23, we find much closer fault-tolerance thresholds of 0.51% and 0.52% for our surface code and the 4.8.8 planar Floquet code, respectively. We note that these decreases from the original thresholds are, respectively, in rough agreement with a 4/5 and 1/2 decrease that one might naïvely anticipate from the syndrome extraction period increases of four to five steps for the surface code and three to six for the 4.8.8 Floquet code, with extra idle noise on introduced for each additional step. For the smallest code size ($d_{\text{f}} = 3$), the pseudo-thresholds are again comparable, though now favoring the surface code at approximately 0.03% for the surface code and 0.02% for 4.8.8 Floquet code. In Fig. 24, we plot the qubit count, circuit depth, and spacetime footprint resource estimates required to achieve logical error rates of $p_{\text{logical}}^{\text{target}} = 10^{-8}$, 10^{-12}, and 10^{-15} for physical error rates in the range $10^{-6} \leq p_{\text{physical}} \leq 10^{-3}$. In Table 3, we list the corresponding fault distances d_{f} required to reach these target logical error rates for physical error rates $p_{\text{physical}} = 10^{-6}$, 10^{-5}, 10^{-4}, and 10^{-3}.

It is worth noting that we anticipate the comparison of our code's performance to that of the 4.8.8 Floquet code to improve when we use a noise model that better reflects the physical errors affecting Majorana hardware. This is due to to natural assumptions, such as two-qubit measurements having higher fault rates than single-qubit measurements, and measurements having higher fault rates than idling error rates; moreover, our code does not utilize $M_{YY:\text{vert}}$ measurements,

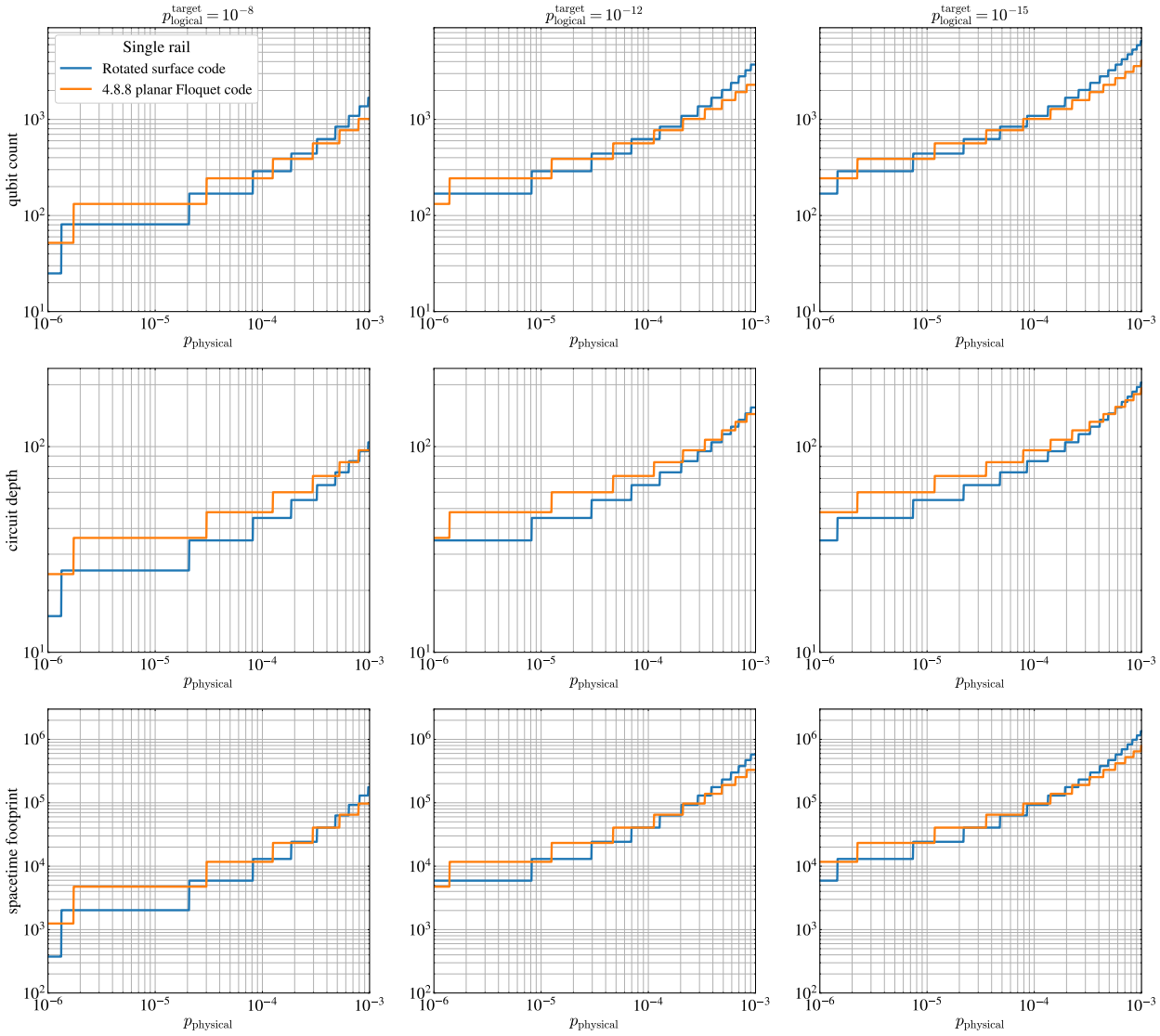


Figure 24: Resource requirements to reach target logical error rates of $p_{\text{logical}}^{\text{target}} = 10^{-8}$, 10^{-12}, and 10^{-15}, comparing single-rail variants of our realization of the surface code and the 4.8.8 planar Floquet code.

	$p_{\text{logical}}^{\text{target}} = 10^{-8}$		$p_{\text{logical}}^{\text{target}} = 10^{-12}$		$p_{\text{logical}}^{\text{target}} = 10^{-15}$	
p_{physical}	SC	4.8.8	SC	4.8.8	SC	4.8.8
10^{-6}	3	3	7	5	7	7
10^{-5}	5	5	9	7	11	9
10^{-4}	9	7	13	11	17	15
10^{-3}	21	15	31	23	41	29

Table 3: Fault distance d_{f} required to reach a target logical error rate $p_{\text{logical}}^{\text{target}}$ of 10^{-8}, 10^{-12} and 10^{-15}, respectively, comparing single-rail variants of our realization of the surface code and the 4.8.8 planar Floquet code. (The total number of qubits is $N = O(4d_{\text{f}}^2)$ for both of these codes.)

which are used in the 4.8.8 Floquet code and can be expected to have higher fault rates than the $M_{XX:\text{horz}}$ and $M_{ZZ:\text{vert}}$ measurements.

As our code provides an interesting test bed for exploring the effect of hook errors, we now investigate this matter in simulation. In Fig. 25, we present the performance for our code (using the original circuits and pipelining of Sec. 2) on a rotated surface code patch with boundary conditions intentionally chosen to align the hook errors with the corresponding logical operators (left) and on a torus (right).[8] The hook errors are malignant for both of these systems. These can be compared to the code performance on a patch with boundary conditions chosen to make hook errors benign, as shown in Fig. 21(left). The choice of boundary conditions should not affect the fault-tolerance threshold, as the circuits implement the same bulk operations. Indeed, the thresholds for the hook-malignant systems in Fig. 25 are estimated to be approximately 0.65% for the planar patch and 0.70% for the torus. The discrepancy between planar and torus is likely due to finite-size effects, and is perhaps not extremely surprising at these sizes.

On the other hand, when hook errors are malignant in the code, it should impact the scaling of the logical failure rate curves in the deep sub-threshold (low p_{physical}) regime, due to the fault distance d_{f} being *halved* as compared to the code distance d. One interesting consequence of this distance-halving is that, in the low-error regime, we expect the curves to "pair up" in terms of slope (when plotted on a log-log scale): the code can correct up to $\lfloor \frac{d_{\text{f}}-1}{2} \rfloor$ faults, with $d_{\text{f}} = \lceil \frac{d}{2} \rceil$, meaning that d must increase by four for the slope to increase by one. Indeed, we observe all of this expected behavior in Fig. 25, where the colored dashed lines represent $p_{\text{logical}} \sim p_{\text{physical}}^{\lfloor (d_{\text{f}}+1)/2 \rfloor}$ scaling, chosen to intercept a reference empirical data point, as previously discussed in the context of Figs. 21 and 23 [see Eq. (11)—although here, the dashed lines are only drawn for visual reference and not used for any subsequent calculations]. More rigorously investigating the deep sub-threshold scaling of these hook-malignant codes with extensive, large-scale *stratified sampling* [5] targeting only the dominant subpopulations expected to contribute to p_{logical} at $p_{\text{physical}} \ll 1$ is an interesting topic for future work. Initial such studies on the hook-malignant rotated surface code patch in Fig. 25(left) indicate that we can indeed at least find fault configurations contributing to p_{logical} at the expected powers

[8] We use p_{logical} to denote the error rate for incorrect recovery of *any* of the logical membranes [14], i.e. which applies for a logical error on either of the two qubits encoded on a torus.

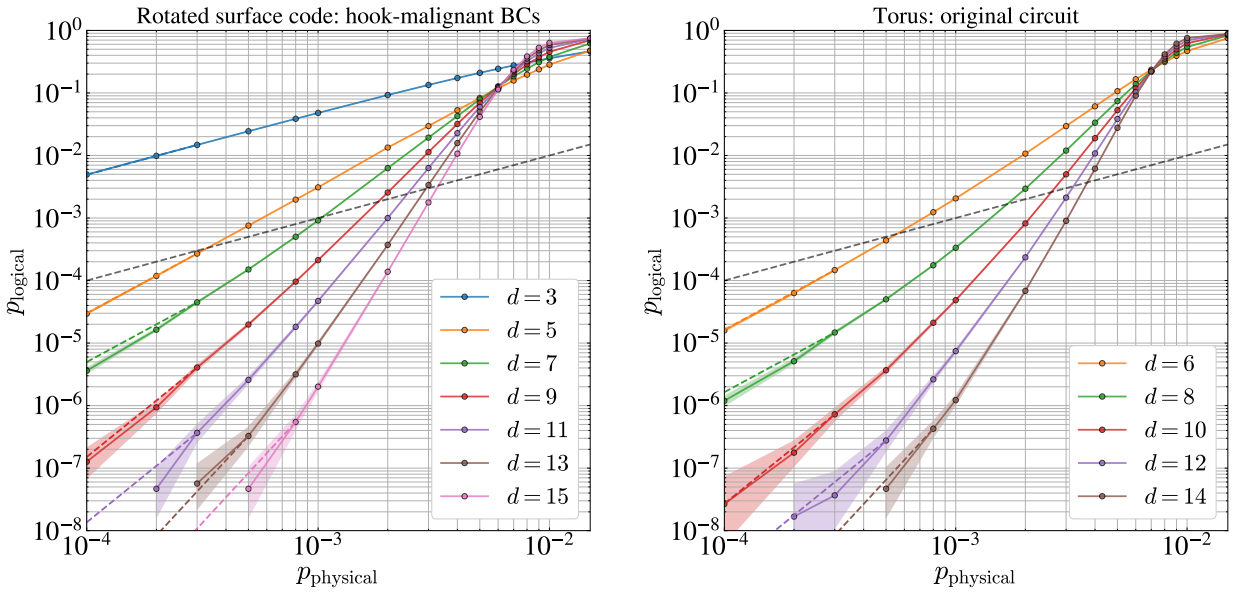


Figure 25: (left) Performance results for our pairwise measurement-based surface code realization on a rotated surface code patch using boundary conditions that make the hook errors malignant (see Sec. 4). We note that the $d = 3$ curve is not expected to intersect with the other curves at (or near) threshold, as the code is not error-correcting at $d = 3$, because the the fault distance is $d_{\text{f}} = 2$. (right) Performance results for our code on a torus. Fault-tolerance thresholds are found to be 0.65% for the planar patch and 0.70% for the torus.

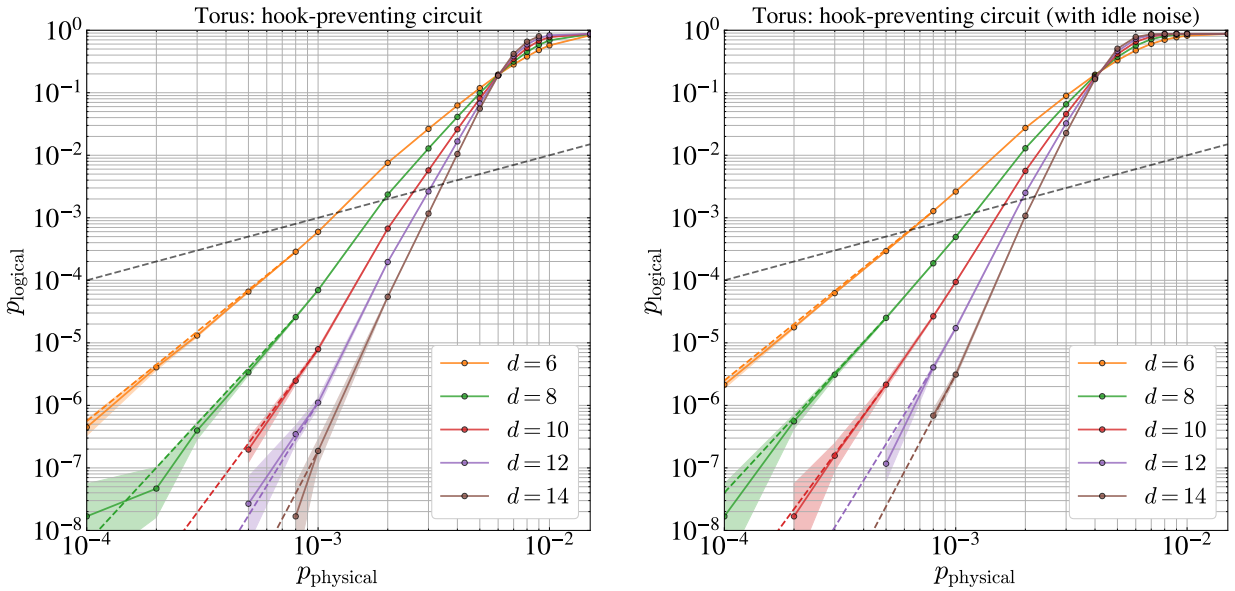


Figure 26: Performance results for the hook-preventing variant of our pairwise measurement-based surface code realization described in Sec. 3 on a torus for noise model without (left) and with (right) idle noise. The full code distance is recovered and performance is improved with respect to the code using the original circuits when hook errors are malignant (see Fig. 25). Fault-tolerance thresholds are found to be 0.61% where there are no idle errors and 0.43% when idle errors are included.

in p_{physical}. For example, at $d = 11$, we can see logical failures for subpopulations contributing to p_{logical} at $O(p^3_{\text{physical}})$, where $3 = \lfloor \frac{\lceil 11/2 \rceil + 1}{2} \rfloor$ is the slope of the dashed curve for $d = 11$ in Fig 25(left). Finally, we remark that hook errors of course have the most dramatic consequence at the smallest $d = 3$, where the code is now no longer even error-correcting in the circuit noise model, and thus we expect $p_{\text{logical}} \propto p_{\text{physical}}$, as observed in the data in Fig. 25(left).

Finally, we evaluate the performance of our code variant utilizing the hook-preventing circuits and pipelining described in Sec. 3. For this, we have performed simulations for the code on a torus for the noise model without and with idle noise. The performance results in Fig. 26 demonstrate the expected recovery of the full distance, i.e. $d_{\text{f}} = d$, in the deep sub-threshold regime. Moreover, the performance is overall better than that of the original circuits when hook errors are malignant. Since the hook-preventing measurement circuits are different from the original circuits, we no longer expect the thresholds to be the same as before. For these hook-preventing variants of our surface code realization on the torus, we find fault-tolerance thresholds of approximately 0.61% when there is no idle noise and 0.43% when there is idle noise. This represents a modest decrease from the threshold value of the code using the original measurement circuits.

Acknowledgments

We are very grateful to A. Paetznick for many useful discussions and help with the decoder implementation and performance assessment. We also thank N. Delfosse for helpful discussions about the decoder construction, M. Beverland for suggesting dynamic weight assignment of decoding graphs for the hook-preventing circuits, J. Weston for assistance setting up the large-scale simulation pipeline on Azure, and A. Paz for providing the parsable circuit format we use to share the simulated circuits. We thank J. Haah, M. Hastings, C. Nayak, and K. Svore for helpful feedback.

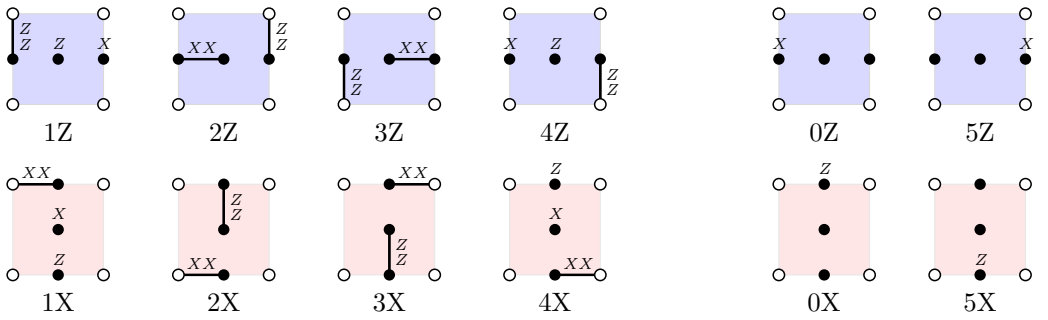


Figure 27: The stabilizer measurement circuits can be interleaved and run on a synchronous schedule, i.e. $(0Z, 0X), (1Z, 1X), \ldots$, by addressing the data qubits in a different order than the pipelined measurement schedule presented in Sec. 2. This interleaved measurement schedule has significant disadvantages compared to the pipelined schedule.

A Interleaved Stabilizer Measurement Circuits

We can interleave M_{ZZZZ} and M_{XXXX} circuits while running them on the same schedule, i.e. $(0Z, 0X), (1Z, 1X), \ldots$, by choosing the order in which data qubits are addressed as shown in Fig. 27. This has the slight benefit of requiring $4r + 2$ steps for r rounds of stabilizer measurement, but it turns out to have significant disadvantages when compared to the pipelined measurement schedule presented in Sec. 2. One disadvantage is that pipelined circuits are expected to perform better than interleaved circuits when using dead component protocols, as discussed in Sec. 5. This is because using interleaved circuits with the dead component protocols will yield fewer total measurements of the superplaquette operators than using pipelined circuits. Another disadvantage arises when using single-rail semiconductor layouts in Majorana hardware, as discussed in Sec. 6. For the interleaved measurement schedule, we find that single-rail layouts would require each circuit step to be split into two steps in order to avoid physically conflicting measurements (overlapping measurement loops). This would double the measurement period to 8 steps, in contrast with the pipelined circuit which could be implemented in single-rail layouts with a 5 step period, introducing greater opportunity for idling errors to damage performance. In terms of code performance, for the best case scenario, i.e. ignoring these dead component and single-rail issues, we find that the interleaved and pipelined scheduling yield very similar performance data, so there is essentially no upside to using the interleaved measurement-based circuits.

B Hook Preventing Modifications for the Pentagonal Tiling Realization of the Surface Code

The pentagonal tiling surface code realization of Ref. [9] utilizes two auxiliary qubits for each 4-gon stabilizer measurement, the circuit of which is shown in Fig. 28. In contrast to our realization, circuit noise for the pentagonal tiling circuits results in bidirectional hook errors, as discussed in Ref. [9]. In particular, for the M_{ZZZZ} circuit, a readout error at the $M_{X_A X_B}$ measurement is equivalent to a $Z_1 Z_3$ or $Z_2 Z_4$ error on the data qubits, while a $Z_A Z_B$ error at the same measurement

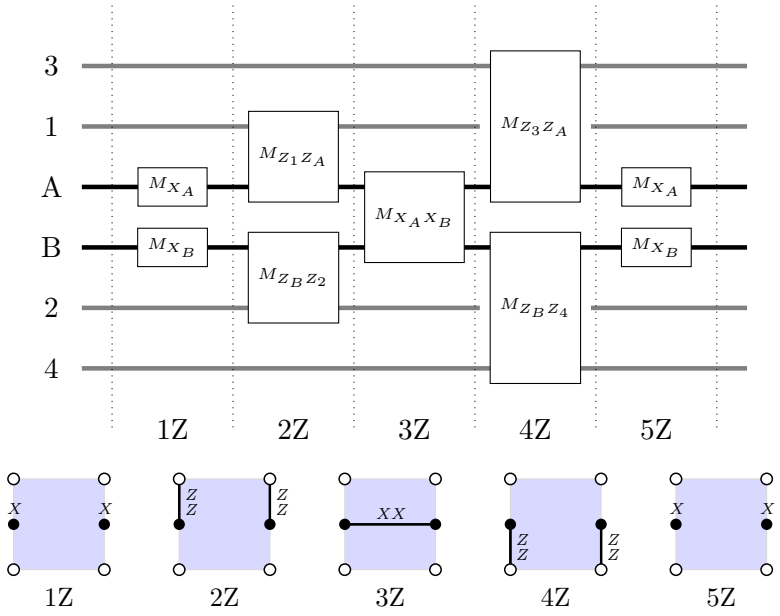


Figure 28: The M_{ZZZZ} circuit from Ref. [9] for the pentagonal tiling realization of the surface code.

is equivalent to a Z_1Z_2 or Z_3Z_4 error on the data qubits. This bidirectionality makes hook errors more problematic for the pentagonal tiling realization. For example, one cannot align these hook errors to be perpendicular to the direction of the corresponding logical operators. Applying our hook-preventing idea to the pentagonal tiling circuit by repeating the pairwise auxiliary qubit measurement, as shown in Fig. 29, will eliminate the hook errors corresponding to the readout errors, though not the two-qubit Pauli errors. The remaining hook errors of our hook-preventing pentagonal tiling circuits are unidirectional with the direction correlated with the error type (though oppositely correlated with our non-hook-preventing circuits): the Z-plaquettes' hook errors correspond to ZZ data qubit errors in the horizontal direction and the X-plaquettes' hook errors correspond to XX data qubit errors in the vertical direction. We note that this is the opposite directionality of hook errors that we found for our stabilizer measurement circuits in Sec. 3. Again, one way of addressing these remaining unidirectional hook errors is to choose logical operators to be aligned perpendicular to the corresponding type of hook errors.

There is another, somewhat more drastic modification one can make to the stabilizer measurement circuits of Ref. [9] that prevents their hook errors in the other direction. Comparing our M_{ZZZZ} circuit in Fig. 2 with the M_{ZZZZ} circuit in Fig. 28, we can retrospectively view our circuit as a modification of the pentagonal tiling M_{ZZZZ} circuit by introducing an additional auxiliary qubit and appropriate measurements to obtain an equivalent circuit. This modification has the effect of trading the horizontal hook error due to a Z_AZ_B error at the pairwise auxiliary qubit measurement in the pentagonal tiling circuit for a vertical hook error due to a Z_B error between the two pairwise auxiliary qubit measurements in our circuit, again leaving only unidirectional hook errors.

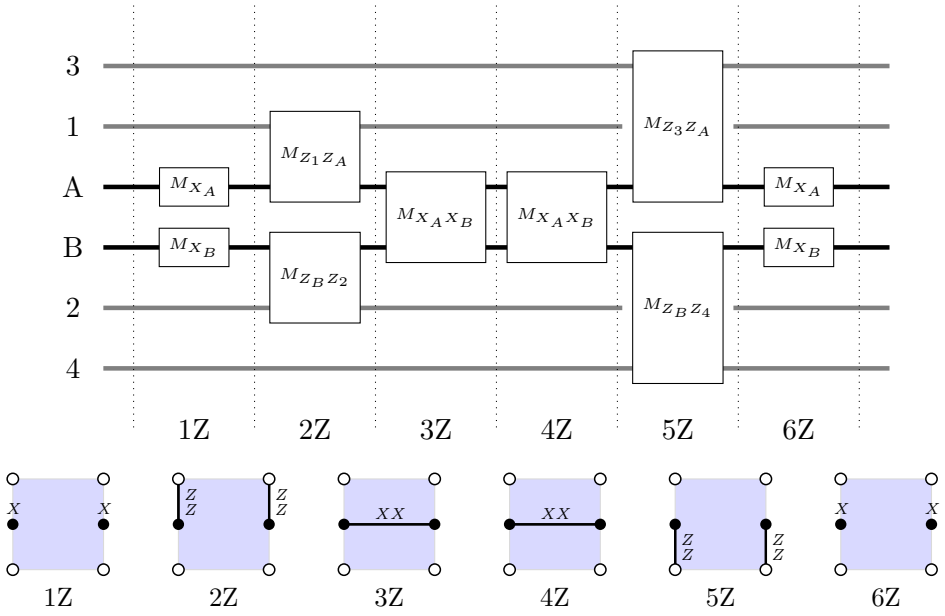


Figure 29: A modification of the M_{ZZZZ} circuit from Ref. [9] that prevents the problematic hook errors associated with readout error at the $M_{X_AX_B}$ measurement. Incorporating this and a similar modification of the M_{XXXX} circuit reduces the problem of bidirectional hook errors to unidirectional hook errors in the pentagonal tiling realization of the surface code.

C Dead Components in Other Surface Code Realizations

In this appendix, we demonstrate our dead components strategy for the measurement-based pentagonal tiling surface code realization of Ref. [9] and the CNOT gate-based realization of the surface code. Step 1 is the same for all realizations, so we can use the modifications shown in Fig. 10 for removing dead data qubits, with the understanding that the array of auxiliary qubits and their collateral removals should be replaced with that of the given realization. For the measurement-based pentagonal tiling realization, the splitting of plaquettes for steps 2 and 3 are shown in Figs. 30 and 31, respectively. For the CNOT gate-based realization, the splitting of plaquettes for steps 2 and 3 are shown in Fig. 32. We note that our strategy may not constitute a desirable trade-off for the CNOT gate-based realization in hardware where measurements could be a prohibitively costly resource, as it would increase in the number of measurements performed for each splitting.

We note that the pentagonal tiling realization of Ref. [9] is pipelined in a manner that exhibits a natural alternation between Z-type and X-type plaquette measurements, similar to our surface code realization. As such, it also has the advantage of measuring superplaquette operators at the maximum rate when using dead component protocols. It is worth mentioning that a similar advantage could potentially be obtained for the CNOT gate-based realization of the surface code by using an appropriate pipelining of the Z-type and X-type circuits. In particular, by offsetting the Z-type and X-type n-gon measurement circuits by three steps (and carefully choosing the order that data qubits are addressed in a circuit), the n-gon measurements effectively alternate between Z-type and X-type. The advantage of this may be undone for hardware in which the measurements time is substantially longer than the CNOT gate time, as measurements will occur

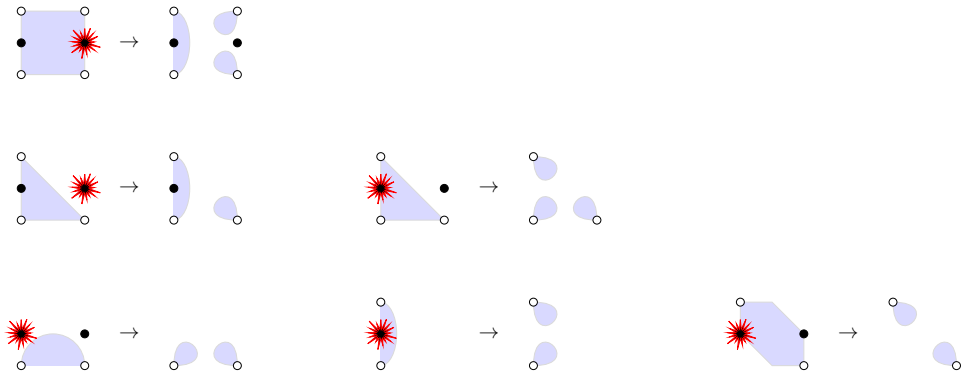


Figure 30: The possible splittings of Z-type n-gons for step 2, where dead auxiliary qubits are removed from the code operation for the measurement-based pentagonal tiling realization of the surface code. (Splittings related to these by rotations and reflections are not shown separately.) The splittings for X-type n-gons may be obtained from these by 90 degree rotations.

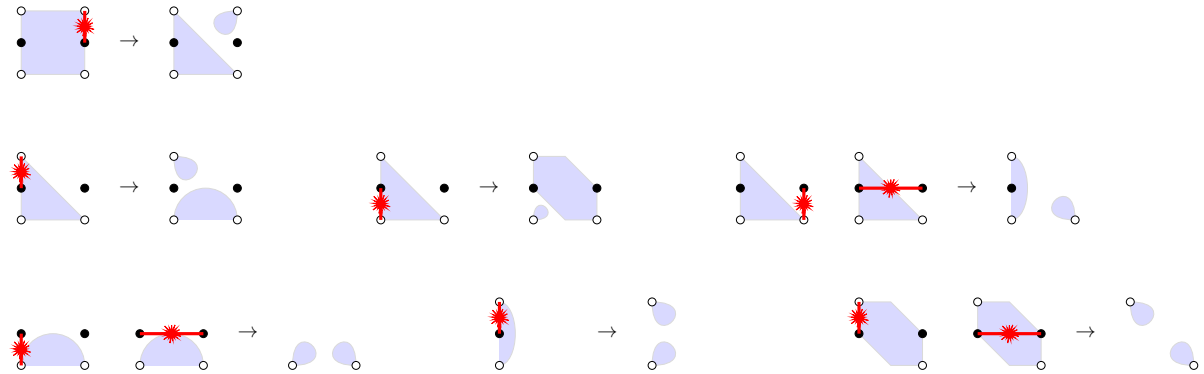


Figure 31: The possible splittings of Z-type n-gons for step 3, where dead connections are removed from the code operation for the measurement-based pentagonal tiling realization of the surface code. (Splittings related to these by rotations and reflections are not shown separately.) The splittings for X-type n-gons may be obtained from these by 90 degree rotations.

Figure 32: The splittings of Z-type or X-type plaquettes for steps 2 and 3, where dead auxiliary qubits and connections are removed from the code operation for the CNOT gate-based realization of the surface code. All possible n-gons splittings are not shown because they all follow the same pattern: for step 2, a dead auxiliary qubit splits the n-gon into n 1-gons; for step 3, a dead connection splits the n-gon into a $(n-1)$-gon and a 1-gon, according to which connection is dead.

during four steps, rather than two steps per cycle with such pipelining.

References

[1] Alexei Yu. Kitaev. "Fault-tolerant quantum computation by anyons". Annals of Physics **303**, 2–30 (2003). arXiv:quant-ph/9707021.

[2] Sergey B. Bravyi and Alexei Yu. Kitaev. "Quantum codes on a lattice with boundary" (1998). arXiv:quant-ph/9811052.

[3] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. "Topological quantum memory". Journal of Mathematical Physics **43**, 4452–4505 (2002). arXiv:quant-ph/0110143.

[4] Michael E. Beverland, Prakash Murali, Matthias Troyer, Krysta M. Svore, Torsten Hoefler, Vadym Kliuchnikov, Guang Hao Low, Mathias Soeken, Aarthi Sundaram, and Alexander Vaschillo. "Assessing requirements to scale to practical quantum advantage" (2022). arXiv:2211.07629.

[5] Adam Paetznick, Christina Knapp, Nicolas Delfosse, Bela Bauer, Jeongwan Haah, Matthew B. Hastings, and Marcus P. da Silva. "Performance of Planar Floquet Codes with Majorana-Based Qubits". PRX Quantum **4**, 010310 (2023). arXiv:2202.11829.

[6] Torsten Karzig, Christina Knapp, Roman M. Lutchyn, Parsa Bonderson, Matthew B. Hastings, Chetan Nayak, Jason Alicea, Karsten Flensberg, Stephan Plugge, Yuval Oreg, Charles M. Marcus, and Michael H. Freedman. "Scalable designs for quasiparticle-poisoning-protected topological quantum computation with Majorana zero modes". Phys. Rev. B **95**, 235305 (2017). arXiv:1610.05289.

[7] Alan Tran, Alex Bocharov, Bela Bauer, and Parsa Bonderson. "Optimizing Clifford gate generation for measurement-only topological quantum computation with Majorana zero modes". SciPost Phys. **8**, 091 (2020). arXiv:1909.03002.

[8] Rui Chao, Michael E. Beverland, Nicolas Delfosse, and Jeongwan Haah. "Optimization of the surface code design for Majorana-based qubits". Quantum **4**, 352 (2020). arXiv:2007.00307.

[9] Craig Gidney. "A Pair Measurement Surface Code on Pentagons" (2022). arXiv:2206.12780.

[10] Matthew B. Hastings and Jeongwan Haah. "Dynamically Generated Logical Qubits". Quantum **5**, 564 (2021). arXiv:2107.02194.

[11] Jeongwan Haah and Matthew B. Hastings. "Boundaries for the Honeycomb Code". Quantum **6**, 693 (2022). arXiv:2110.09545.

[12] Craig Gidney, Michael Newman, and Matt McEwen. "Benchmarking the Planar Honeycomb Code". Quantum **6**, 813 (2022). arXiv:2202.11845.

[13] John van de Wetering. "ZX-calculus for the working quantum computer scientist" (2020). arXiv:2012.13966.

[14] Héctor Bombín, Chris Dawson, Ryan V. Mishmash, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. "Logical Blocks for Fault-Tolerant Topological Quantum Computation". PRX Quantum **4**, 020303 (2023). arXiv:2112.12160.

[15] Héctor Bombín and Miguel A. Martin-Delgado. "Optimal resources for topological two-dimensional stabilizer codes: Comparative study". Phys. Rev. A **76**, 012305 (2007). arXiv:quant-ph/0703272.

[16] Yu Tomita and Krysta M. Svore. "Low-distance surface codes under realistic quantum noise". Phys. Rev. A **90**, 062320 (2014). arXiv:1404.3747.

[17] James M. Auger, Hussain Anwar, Mercedes Gimeno-Segovia, Thomas M. Stace, and Dan E. Browne. "Fault-tolerance thresholds for the surface code with fabrication errors". Phys. Rev. A **96**, 042316 (2017). arXiv:1706.04912.

[18] Christopher Chamberland and Michael E. Beverland. "Flag fault-tolerant error correction with arbitrary distance codes". Quantum **2**, 53 (2018). arXiv:1708.02246.

[19] Craig Gidney. "Stim: a fast stabilizer circuit simulator". Quantum **5**, 497 (2021). arXiv:2103.02202.

[20] Thomas M. Stace, Sean D. Barrett, and Andrew C. Doherty. "Thresholds for Topological Codes in the Presence of Loss". Phys. Rev. Lett. **102**, 200501 (2009). arXiv:0904.3556.

[21] Armands Strikis, Simon C. Benjamin, and Benjamin J. Brown. "Quantum Computing is Scalable on a Planar Array of Qubits with Fabrication Defects". Phys. Rev. Applied **19**, 064081 (2023). arXiv:2111.06432.

[22] Oscar Higgott and Craig Gidney. "Sparse Blossom: correcting a million errors per core second with minimum-weight matching" (2023). arXiv:2303.15933.

[23] Nicolas Delfosse and Adam Paetznick. "Spacetime codes of Clifford circuits" (2023). arXiv:2304.05943.

[24] Nicolas Delfosse, Adam Paetznick, Jeongwan Haah, and Mathew B. Hastings. "Splitting decoders for correcting hypergraph faults" (2023). arXiv:2309.15354.

[25] Richard M. Karp. "Reducibility among combinatorial problems". Pages 85–103. Springer US. Boston, MA (1972).

[26] Christopher A. Pattison, Michael E. Beverland, Marcus P. da Silva, and Nicolas Delfosse. "Improved quantum error correction using soft information" (2021). arXiv:2107.13589.

[27] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. "Surface codes: Towards practical large-scale quantum computation". Phys. Rev. A **86**, 032324 (2012). arXiv:1208.0928.